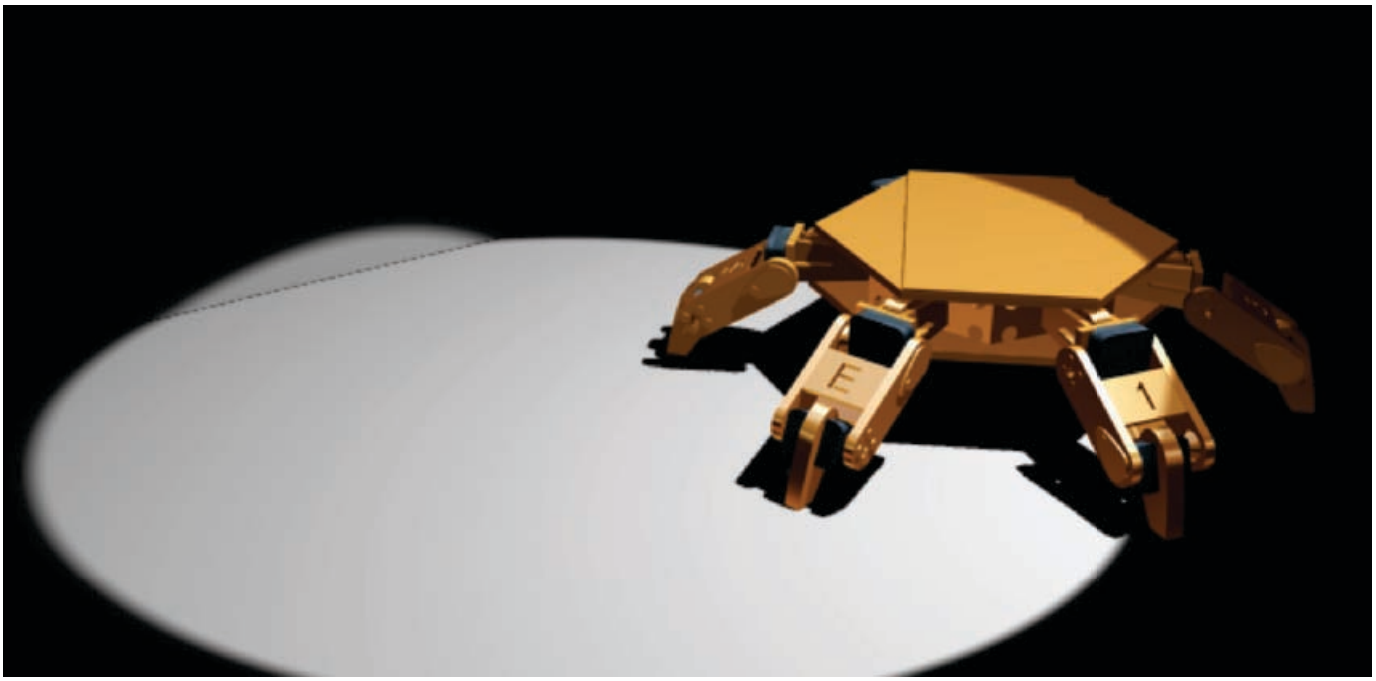
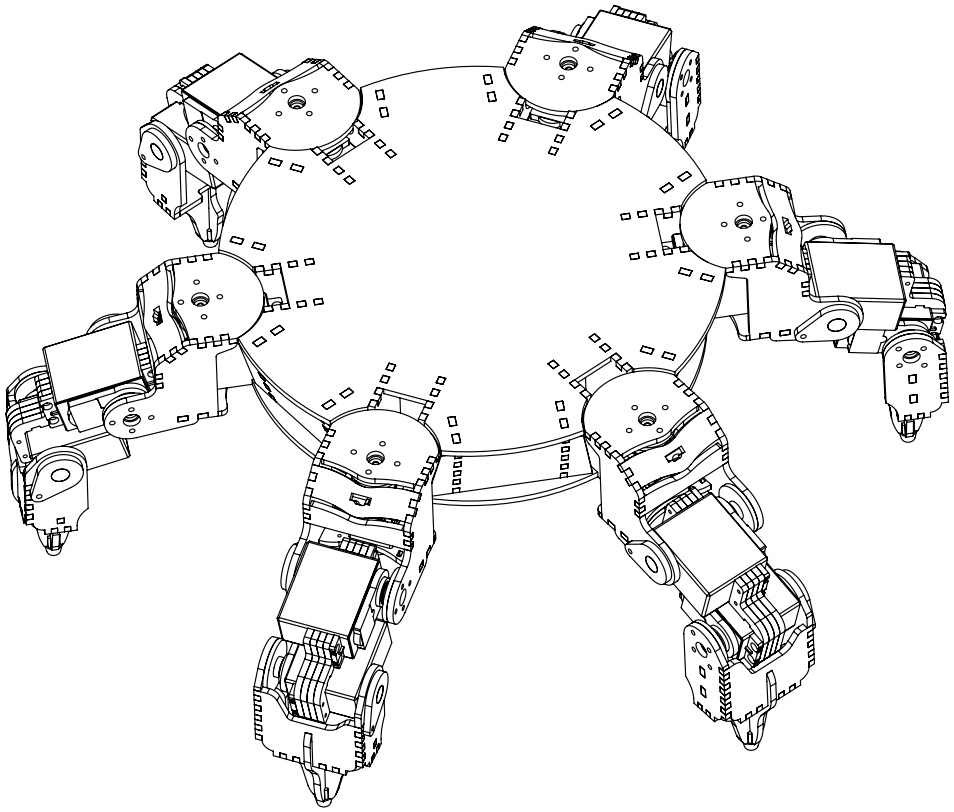
HSSE-SandboxS 2 – Handbuch
Markus Pfaff, Michael Kreuzgruber, Christian Raschko,
Michael Bogner, Armin Brandl
2. Auflage, © 20. September 2007, SVN: 408



HSSE-SandboxS 2
Handbuch



Nutzungsbedingungen Die Konstruktion der „SandboxS 2“, alle Abbildungen, Konstruktionszeichnungen, Fotos und der Text dieser Beschreibung stehen unter dem *Copyright* der jeweiligen Autoren. Ihre weltweite Nutzung zu nichtkommerziellen Zwecken innerhalb der Ausbildung (in Schulen und ähnlichen Ausbildungsstätten ebenso wie im Ausbildungsbereich von Firmen) und im Privatbereich wird bis auf (auch in Einzelfällen möglichen) Widerruf gestattet. Bei solcher Nutzung ist an einer, z.B. für Zuschauer einer Veranstaltung gut sichtbaren Stelle auf den Ursprung der „SandboxS 2“ hinzuweisen. Im Einzelnen besteht dieser Hinweis aus dem Logo der FH-OÖ, dem Schriftzug „HSSE“ als Hinweis auf den Studiengang Hardware/Software Systems Engineering und dem Schriftzug „WWW.SANDBOX.AT“. Unterhält der Nutzer (Schule, Firma, Privatperson) WWW-Seiten, so ist ein *Link* auf `www.sandbox.at` dort vorzusehen. Eine kommerzielle Nutzung ist nur mit schriftlicher Genehmigung und Lizenz der Autoren statthaft.

Haftungsausschluss Bitte lesen Sie die am Rand mit gekennzeichneten Textteile besonders aufmerksam und beherzigen Sie die darin enthaltenen Warnungen. Bitte beachten Sie, dass die Ausführung sämtlicher Anweisungen in dieser Anleitung Ihrem eigenen Risiko obliegt. Die „SandboxS 2“ stellt kein Produkt dar! Es handelt sich insofern lediglich um Empfehlungen, die zwar weitgehend erprobt, aber möglicherweise nicht fehlerfrei sind. Alle Anweisungen richten sich ausschließlich an vollrechtsfähige Personen (die demnach auch im Sinne des Gesetzes erwachsen sein müssen). Sollen Anweisungen in dieser Anleitung von anderen als solchen Personen ausgeführt werden, so sollte dies nur unter Aufsicht einer fachkundigen und vollrechtsfähigen Person geschehen, welche diese Aufsicht auszuüben berechtigt ist. Weder der Autor, noch die FH-Oberösterreich Studienbetriebsgesellschaft übernehmen eine Haftung für Schäden, die durch Befolgung oder auch Nichtbefolgung dieser Anweisung entstehen. Ebensowenig wird eine Haftung für Schäden im Zusammenhang mit dem Betrieb der in dieser Anleitung beschriebenen Platine übernommen. Für den Inhalt verlinkter WWW-Seiten und den Inhalt von empfohlener Literatur wird keine Verantwortung übernommen. Sollten solche Seiten im Laufe der Zeit problematische Inhalte bekommen, sind wir für einen Hinweis dankbar.

Zeichenlegende Einige Textpassagen sind am Textrand mit Symbolen folgender Bedeutung gekennzeichnet:

Gefahr! Bei Nichtbeachtung besteht entweder Verletzungs- oder zumindest Zerstörungsgefahr!

Hinweis Hinweis, den man beherzigen sollte. Es besteht zwar keine unmittelbare Gefahr, wenn man dies nicht tut, aber es wird beispielsweise eine Funktion beeinträchtigt.

Tipp Hinweis, dessen Beachtung meist Vorteile bringen wird.

Vorwort

Als ein schwarzer Hexapodroboter in Hagenberg im Sommer 2003 im Rahmen eines einsemestrigen Studienprojektes des Jahrgangs 2001 von HSSE seine ersten noch etwas zaghaften Gehversuche unternahm, ahnte man noch nichts davon, mit welcher Geschwindigkeit sich diese „Tierchen“ quasi wie von selbst weiterentwickeln würden. Denn nach diesem ersten Erfolg wurde die Weiterentwicklung zunächst einmal auf Grund der anderweitigen Projekte und Kooperationen mit der Industrie abrupt gestoppt.

Nach einjähriger Pause fand sich im Herbst 2004 dann aber in einem Jahresstudienprojekt des HSSE-Jahrgangs 2002 wieder eine StudentInnengruppe mit dem Ziel einer Weiterentwicklung des sechsbeinigen Roboters zusammen. Das Erstlingsmodell aus dem Vorjahr litt an drei Kernproblemen, die von der Gruppe systematisch angegangen wurden:

a) Seine Größe bedingte leider auch große Probleme mit den Servoantrieben. Die Gelenke waren nur von geringer Lebensdauer. Die Umkonstruktion in Verbindung mit einer erheblichen Verkleinerung führten dann auch zum Roboter „Die wilde 13“. Dieser Roboter konnte nun auch auf der hauseigenen CNC-Fräse gefertigt werden.

b) Die ursprünglich verwendeten Gehmuster waren von den Vorbildern aus der Biologie weit entfernt. Sie entstanden hauptsächlich durch immer weiter getriebene Verfeinerung eines mittels Tabelle fest programmierten Ablaufs, mit anderen Worten: Durch Probieren. An Kurvengang war beispielsweise ursprünglich nicht zu denken. Nun wurden stattdessen von Insekten abgeschaut, während die Bewegungssteuerung auf die Prinzipien der Industrierobotik zurückgriff, allen voran die inverse Kinematik.

c) Die ursprüngliche Steuerungselektronik war ein speziell für den Zweck der Servosteuerung umgebautes und erweitertes Evaluierungsboard für den Chipentwurf, die Sandbox. Die Steuerung wurde daher auch als reiner Chipentwurf in einem FPGA des Herstellers Xilinx verwirklicht. Diesem reinen Hardwareansatz fehlte es naturgemäß an der Flexibilität des Softwareanteils. Der enorme Vorteil der extrem genauen zeitlichen Steuerung gegenüber den Möglichkeiten eines Systems auf Basis eines Mikrocontrollers war andererseits offensichtlich. Als Konsequenz hieraus wurde die SandboxS entwickelt. Hier wurde ein ALtera-FPGA mit einem AVR-Mikrocontroller gekoppelt. Damit war die Steuerung nun ein Hardware/Software-Co-Design.

Die CNC-Fräse der FH in Hagenberg ermöglichte nicht nur die genaue Fertigung eines einzelnen Prototypen, sondern mit ihrer Hilfe wäre sogar die Auflage einer Kleinstserie möglich. Die Steuerungselektronik SandboxS war von vornherein nach den Gesichtspunkten einer möglichen Serienproduktion konstruiert. Der Gedanke, den Hexapodroboter als Botschafter in Schulen zu schicken, um die Kooperation insbesondere der HTBLAs und AHS mit technischer Orientierung mit dem Studiengang zu fördern, lag nahe. Dies war die Geburtsstunde der österreichischen Hexapod-Meisterschaften, die mittlerweile alljährlich im April in Hagenberg ausgetragen werden und sich in der Robotikszene einen festen Platz erobert haben.

Das vorliegende Handbuch beschreibt die Weiterentwicklung der Steuerungselektronik SandboxS, die SandboxS 2. Der ursprünglich verwendete Mikrocontroller wurde durch einen Typen ersetzt, der eine eingebaute USB-Schnittstelle aufweist und das FPGA stammt nun vom Hersteller Lattice und besitzt einen Kon gurationsspeicher auf Flash-Basis. Der Betrieb erfolgt in dieser Version aus umweltfreundlichen NiMh-Akkus, die einfach in Akkuhalter auf der Platine eingesetzt werden können.

Kern der SandboxS 2 ist nach wie vor der HSSE-Robotics-Chip, der gegenüber der ursprünglichen Version einen enorm erweiterten Funktionsumfang anbietet und in der neuesten Version sogar einen CORDIC-Koprozessor zur schnellen Berechnung der inversen Kinematik und der für die Robotik typischen Koordinatentransformationen (Givens-Rotation) enthält. In der Software wird der Zugang zu dieser Vielzahl an komplexen Funktionen durch eine Funktionsbibliothek für die Programmierung des AVR-Mikrocontrollers in C mittlerweile ebenfalls sehr erleichtert.

Auch auf der mechanischen Seite konnten wir mit dem neuen Roboterentwurf „Captain Ahab“ einen Fortschritt verzeichnen. Die enge Zusammenarbeit des Studiengangs mit der Firma trotec aus Wels auf dem Gebiet der Lasertechnik, ermöglicht es nun, einen Lasercutter zum Ausschneiden der Roboterbauteile einzusetzen. Die erreichbare Genauigkeit liegt im 1/100-mm-Bereich.

Das Hexapod-Projekt der Studiengänge HSSE und ESD wäre ohne die enge Zusammenarbeit von Studierenden und Professoren in dieser Form nicht denkbar. Unter anderem schlägt sich diese Zusammenarbeit in diesem mittlerweile zu stattlichem Umfang gewachsenen Handbuch nieder, wie die Zusammensetzung des Autorenteams erkennen lässt.

Ein Dank an dieser Stelle den industriellen Sponsoren des Projektes: technosert/Wartberg, trotec/Wels, Atmel/Heilbronn, Hitec-Robotics/Bretten, Aigner-Balsa/Grünburg und MSC/Wien.

Hagenberg im September 2007

Inhaltsverzeichnis

1 Überblick **11**
1.1 Merkmale . . . 12

2 Erste Schritte **14**
2.1 Systemvoraussetzungen . . . 14
2.2 Entwicklungsumgebung einrichten . . . 14
2.2.1 WinAVR installieren . . . 15
2.2.2 Atmel AVR Studio installieren . . . 17
2.3 Gerätetreiber installieren . . . 19
2.3.1 JTAG ICE Mk II Treiber installieren . . . 19
2.3.2 SandboxS 2 Treiber installieren . . . 21
2.4 „Hello World“ in der Sandbox-Version . . . 24
2.4.1 Projekt erstellen . . . 24
2.4.2 Programm schreiben . . . 29
2.4.3 Projekt übersetzen . . . 31
2.4.4 Programm ausführen . . . 32
2.4.5 Programm testen . . . 37
2.4.6 Zusammenfassung . . . 38
2.4.7 Aufgaben . . . 38
2.5 Laufende Lichter . . . 38
2.5.1 Projekt erstellen . . . 38
2.5.2 Programm schreiben . . . 38
2.5.3 Projekt übersetzen . . . 41
2.5.4 Programm ausführen . . . 41
2.5.5 Programm testen . . . 41
2.5.6 Zusammenfassung . . . 41
2.5.7 Aufgaben . . . 41
2.6 Servokalibrierung mit Winkelzeiger . . . 41
2.6.1 Projekt erstellen . . . 41
2.6.2 Programm schreiben . . . 42
2.6.3 Projekt übersetzen . . . 46
2.6.4 Programm ausführen . . . 46
2.6.5 Programm testen . . . 46
2.6.6 Zusammenfassung . . . 46
2.6.7 Aufgaben . . . 46

3 Energieversorgung **48**
3.1 Akkuzellen . . . 49
3.2 Ladegerät . . . 49

3.3 Akkuhalter . . . 49
3.4 Ein-/Ausschalter . . . 50
3.5 Entladeschaltung . . . 51
3.6 Verpolungsschutzschaltung . . . 52
3.7 Spannungsregelung . . . 52
3.7.1 Spannungsversorgung der Elektronikkomponenten . . . 52
3.7.2 Spannungsversorgung RC-Empfänger . . . 53
3.7.3 Spannungsversorgung PS/2-Buchse . . . 53
3.7.4 Spannungsversorgung des ADCs im Mikrocontroller . . . 53

4 Mikrocontroller Atmel AVR AT90USB1286/7 55
4.1 Emulation mit dem JTAG ICE Mk II . . . 55
4.2 Taktversorgung . . . 55
4.3 Reset . . . 57
4.4 Interrupts . . . 57
4.5 External Memory Interface . . . 57
4.6 Einstellung der Fuses des AVR . . . 57
4.7 Update der Software des AVR-Mikrocontrollers . . . 58
4.7.1 USB-Bootloader . . . 58
4.7.2 Software-Update mittels Atmel AVR-Studio und JTAG ICE Mk II . . . 61
4.8 Atmel AVR Studio und WinAvr . . . 62

5 On-Chip Peripherie des Atmel AVR AT90USB1286/7 64
5.1 USB-Function Controller . . . 64
5.2 SD-Card . . . 65
5.3 Stromaufnahme und Akkuspannung . . . 69
5.3.1 Strommessung . . . 69
5.3.2 Spannungsmessung . . . 70
5.4 Taster . . . 70
5.5 Status-LED . . . 70
5.6 Erweiterungsstecker . . . 71
5.7 Anschluss eines Bluetooth-RS323-Moduls . . . 72

6 FPGA Lattice XP LFXP3K 74
6.1 Taktversorgung . . . 74
6.2 Reset . . . 74
6.2.1 Hardware-Bootloader-Aufruf des AVRs . . . 74
6.3 FPGA-Kon guration . . . 75
6.3.1 Update der FPGA-Kon guration . . . 75

7 Robotik-Peripherie: HSSE-Robotics-Chip 81
7.1 Peripheriekomponenten aus Sicht des Mikrocontrollers . . . 82
7.1.1 Zugriff auf 16-Bit-Register . . . 83
7.2 Interrupts . . . 83
7.2.1 Interrupt-Flag-Register . . . 85
7.3 Servosteuerung . . . 87

7.4 Servo-Anschlüsse . . . 92
7.5 Servopulsgenerator . . . 92
7.5.1 Mode . . . 92
7.5.2 Pulsunterdrückung . . . 92
7.5.3 Drehrichtungsumkehr . . . 94
7.5.4 Pulsdauer . . . 95
7.5.5 Pulswiederholungszyklus . . . 96
7.6 Servolinearisierung und -kalibrierung . . . 96
7.6.1 Vorgehensweise für die Servokalibrierung . . . 99
7.7 Servo-Sequencer . . . 102
7.7.1 Geschwindigkeitsabstimmung . . . 106
7.8 Servogrundstellungsmodus . . . 107
7.9 Servopaniktaste . . . 107
7.10 LED mit Helligkeitssteuerung . . . 108
7.11 Status-LED . . . 109
7.12 Taster . . . 109
7.13 RC-Empfängerdecoder . . . 110
7.13.1 RC-Empfängeranschluss . . . 110
7.14 PS/2-Mausinterface . . . 113
7.15 Piezosummer mit Tongenerator . . . 114
7.16 Erweiterungssteckplatz . . . 115
7.17 Sony-Playstation 2 Game-Pad-Controller . . . 116
7.17.1 Adapterkabel . . . 117
7.18 Globaler Timer . . . 122
7.19 Versionsnummer des Robotics-Chips . . . 123
7.20 Demo-Mode . . . 123
7.21 RAM-Erweiterung . . . 123
7.22 CORDIC-Koprozessor . . . 124
7.22.1 Zahlenformate, Genauigkeit . . . 126
7.23 Adressraum im Überblick . . . 130

8 Inverse Kinematik 133
8.1 Allgemeines . . . 133
8.1.1 Vorwärtskinematik . . . 133
8.1.2 Inverse Kinematik . . . 133
8.2 Inverse Kinematik des Hexapods . . . 134
8.3 Hexapod Kalibrierung . . . 135
8.3.1 Spezi kation . . . 135
8.3.2 Messung . . . 136
8.3.3 Quellcode . . . 138
8.4 Inverse Kinematik Test . . . 142
8.4.1 Spezi kation . . . 142
8.4.2 Quellcode . . . 143

9 Software-Funktionsbibliothek 146
9.1 Übersicht der Schnittstellen . . . 146

9.2 Datentypen . . . 150
9.3 Initialisierung . . . 150
9.4 SandboxS2_lib . . . 150
9.4.1 SL2_Init . . . 151
9.4.2 SL2_AvrSetMsgLed . . . 151
9.4.3 SL2_AvrGetKey . . . 152
9.4.4 SL2_AvrWaitOnKey . . . 153
9.4.5 SL2_ExtAvrSetPinMode . . . 154
9.4.6 SL2_ExtAvrGetPinValue . . . 155
9.4.7 SL2_ExtAvrSetPinValue . . . 155
9.4.8 SL2_GetCurrent . . . 156
9.4.9 SL2_GetCurrentLowAmpli cation . . . 157
9.4.10 SL2_GetCurrentHighAmpli cation . . . 158
9.4.11 SL2_GetBatteryVoltage . . . 158
9.4.12 SL2_ExtAvrGetAdcValue . . . 159
9.5 Schnittstellen und Datenströme (USB und UART) . . . 160
9.6 UART . . . 162
9.6.1 uart_init . . . 162
9.6.2 uart_init_double_speed . . . 164
9.6.3 uart_putc . . . 164
9.6.4 uart_puts . . . 165
9.6.5 uart_getc . . . 165
9.6.6 uart_is_rx_empty . . . 166
9.6.7 uart_is_tx_full . . . 166
9.7 USB-UART . . . 167
9.7.1 uart_usb_init . . . 167
9.7.2 uart_usb_putchar . . . 169
9.7.3 uart_usb_getchar . . . 169
9.7.4 uart_usb_test_hit . . . 170
9.7.5 uart_usb_tx_ready . . . 170
9.7.6 uart_usb_flush . . . 171
9.8 RoboticsChip . . . 171
9.8.1 RC_Init . . . 171
9.8.2 RC_SetLedBrightness . . . 172
9.8.3 RC_SetLedClearAll . . . 173
9.8.4 RC_SetServoMode . . . 173
9.8.5 RC_SetGlobalServoMode . . . 175
9.8.6 RC_SetServoPwmDirect . . . 176
9.8.7 RC_GetServoCurrentPwm . . . 177
9.8.8 RC_SetServoPhiSpeed . . . 178
9.8.9 RC_SetServoPhiDirect . . . 179
9.8.10 RC_GetServoDeltasToGo . . . 181
9.8.11 RC_GetServoCurrentDeltaPhi . . . 182
9.8.12 RC_GetServoCurrentPhi . . . 183
9.8.13 RC_SetServoRotation . . . 185
9.8.14 RC_SetServoAttributesTable . . . 186

9.8.15 RC_SetServoAttributesTable_P . . . 186
9.8.16 RC_SetServoAttributesTable_E . . . 186
9.8.17 RC_SetGlobalServoAttributesTable . . . 186
9.8.18 RC_SetGlobalServoAttributesTable_P . . . 187
9.8.19 RC_SetGlobalServoAttributesTable_E . . . 187
9.8.20 RC_SetServoTuneFine . . . 187
9.8.21 RC_SetServoTuneCoarse . . . 188
9.8.22 RC_GetPs2Data . . . 189
9.8.23 RC_SetRxControl . . . 190
9.8.24 RC_GetRxData . . . 191
9.8.25 RC_GetGamePadButtons . . . 192
9.8.26 RC_GetGamePadJoystickData . . . 193
9.8.27 RC_GetGamePadData . . . 194
9.8.28 RC_SetBuzzerVolume . . . 195
9.8.29 RC_SetBuzzerPhaseHigh . . . 195
9.8.30 RC_SetBuzzerPhaseLow . . . 196
9.8.31 RC_SetCordicCalculationArguments . . . 197
9.8.32 RC_SetCordicCalculationStart . . . 198
9.8.33 RC_CordicWaitOnResult . . . 198
9.8.34 RC_GetCordicCalculationResult . . . 199
9.8.35 RC_GetRoboticsChipVersion . . . 200
9.8.36 RC_GetGlobalTimer . . . 200
9.8.37 RC_SetInterrupt0 . . . 201
9.9 Inverse Kinematik . . . 202
9.9.1 IK_Init . . . 202
9.9.2 IK_CalculateInverseKinematicsCoxa . . . 203
9.9.3 IK_SetLegMovementCoxa . . . 205
9.9.4 IK_SetLegMovement . . . 206
9.9.5 IK_GivensTransformation . . . 208
9.10 SD-Karte . . . 209
9.10.1 mmc_init . . . 209
9.10.2 fat_cluster_data_store . . . 210
9.10.3 GetClusterSize . . . 210
9.10.4 GetBytesPerSec . . . 211
9.10.5 fat_search_ le . . . 212
9.10.6 fat_read_ le . . . 213
9.10.7 fat_write_ le . . . 214
9.11 Strukturen und benutzerde nierte Typdeklarationen . . . 216
9.11.1 Robotics-Chip . . . 216
9.11.2 Inverse Kinematik . . . 217

A Schaltplan **219**

B Platinenlayout **224**
B.1 Platinenfoto ohne Bestückungsdruck . . . 224
B.2 Platinenfoto ohne Bestückungsdruck . . . 225

B.3 Platinenfoto mit Aufklebern 225
B.4 Platinenfoto mit Aufklebern und Akkuzellen 226
B.5 Foto der Platinenunterseite 226
B.6 Bestückungsaufdruck 227
B.7 Aufkleber statt Bestückungsaufdruck 228
B.8 Layout Top-Layer 229
B.9 Layout Bottom-Layer 230

C RC-Empfängerumbau zum Anschluss an die SandboxS 2 231

D Literaturverzeichnis 235

Index 235

1 Überblick

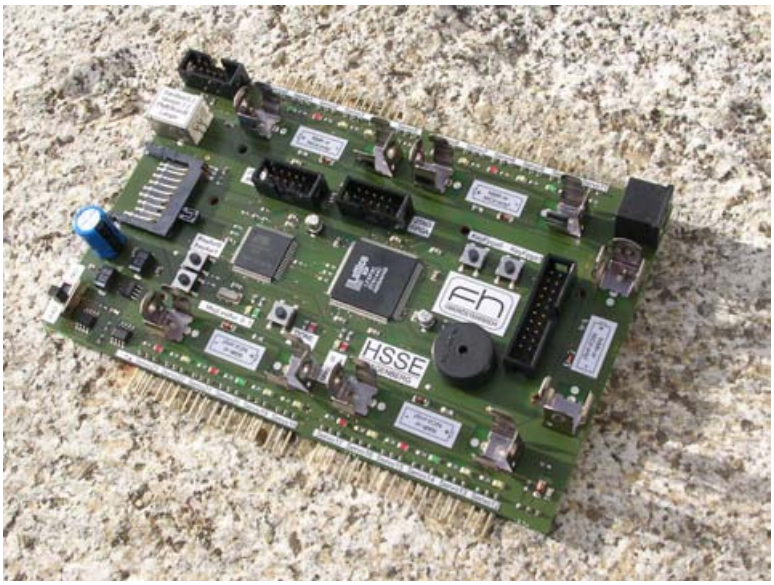


Abbildung 1.1: Die SandboxS 2.

Das *Embedded System* SandboxS 2 (Abb. 1.1, links) ist eine frei kon gurierbare Steuerelektronik für bis zu 24 Servoantriebe. Bei der Entwicklung stand das Feld der *Embedded Robotics* als wichtigster Anwendungbereich im Vordergrund. Als Anwendungsbeispiel seien die Hexapod-Laufroboter „Wilde 13“ und „Captain Ahab“ genannt, die in Abbildung 1.2 zu sehen sind. Das System sollte darüber

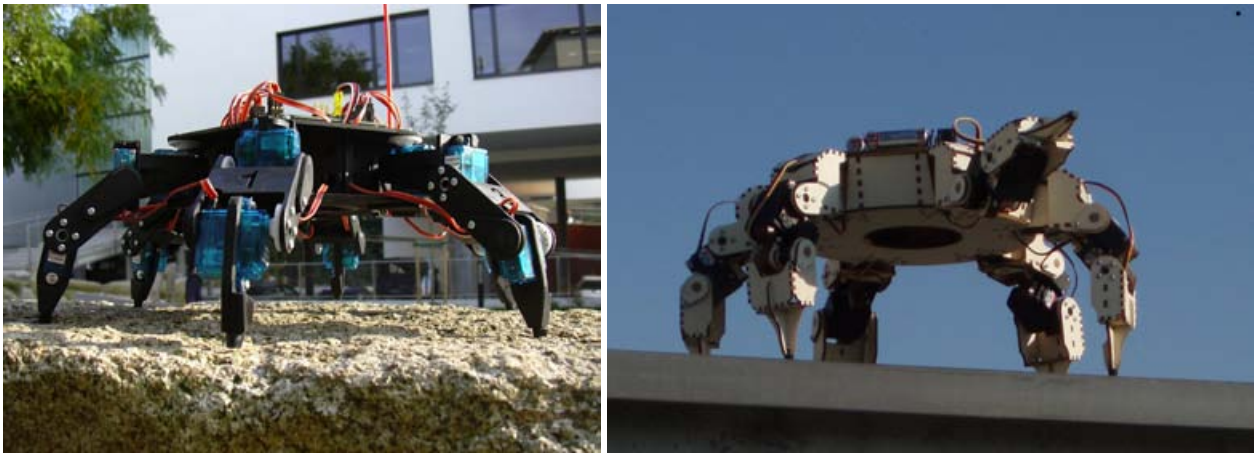


Abbildung 1.2: Zwei Hexapods vor ihrem Zuhause in Hagenberg. Links: Eine *Wilde 13* (Pa13), Rechts: Die wesentlich größere Konstruktion *Captian Ahab* (Bo20).

hinaus auch für Roboter verwendbar sein, wie sie Abb. 1.3 zeigt.

Abbildung 1.3: Mittlere bis große Octapods stecken die obere Grenze des Einsatzspektrums der SandboxS 2 ab. Die Aufnahmen entstanden im Laboratorium von HSSE und bei einer Exkursion auf dem Versuchsfreigelände in Hagenberg/Mühlkreis.

Die SandboxS 2 eignet sich über die Robotik hinaus auch für die Bewegungssteuerung animierter Objekte und die Maschinensteuerung. Die Bewegung kann synchron zu Video- und Audiodaten erfolgen, wobei MIDI-Daten per USB-Schnittstelle übertragen werden können.

Die SandboxS 2 ist ein offenes Projekt der Studiengänge Hardware Software Systems Engineering (HSSE) und Embedded Systems Design (ESD) an der FH-OÖ/Hagenberg. Die SandboxS 2 wird von der Firma *technosert* in Wartberg/Aist gefertigt und nanziell unterstützt.

1.1 Merkmale

- Software auf Basis des Mikrocontrollers Atmel AVR At90USB1286/7
  - Kostenloser OpenSource-C-Compiler WinAVR
  - Kostenloser Simulator AvrStudio von Atmel
  - Preisgünstiger Emulator AVR JTAG ICE Mk II
  - Kostenloser USB-Bootloader zur Flash- und EEPROM-Programmierung
  - Umfangreiche Software-Funktionsbibliothek
- Robotik-*Peripherals* des HSSE-Robotik-Chip V2
  - Implementiert in einem FPGA der Lattice-XP-Familie
  - Umfangreiche Kontrollfunktionen für bis zu 24 handelsübliche Modellbauservos
  - Parametrisierbare Linearisierung des Zusammenhangs von Soll- und Istwinkel für jedes einzelne Servo
  - Programmierbare Servo-Bewegungssequenzen
  - Servogrundstellung und -deaktivierung auf Tastendruck
  - Speziell an die Erfordernisse der Robotik angepasste Timerfunktionen

  - Auswertung des Signals eines RC-Empfängers mit bis zu 12 Kanälen
  - 24 LEDs mit in 256 Stufen einstellbarer Helligkeit
  - PS/2-Mausschnittstelle
  - Schnittstelle für einen kabelgebundenen oder einen drahtlosen Sony Playstation-2-Game-Pad-Controller
  - Programmierbare Auslösung von IRQs als Reaktion auf bestimmte Ereignisse
  - Tastenkombination startet AVR-Bootloader nach Reset
  - CORDIC-Koprozessor, beispielsweise zur Berechnung der inversen Kinematik
- USB-*Interface*
  - USB 2.0 Full-Speed Function
  - *Bootloader* für AVR-Mikrocontroller ermöglicht *Update* und Basisfunktionalität zum *Debugging* wenn auf JTAG ICE Mk II verzichtet werden soll
  - MIDI-Schnittstellenemulation über USB möglich
  - USB-Kommunikation nach CDC-Protokoll (*Communication Device Class*) ist Bestandteil der SW-Bibliothek
- SD-Card-Slot
  - FAT-Dateisystem auf SD-Karte wird von SW-Bibliothek unterstützt
- Energieversorgung
  - Akkubetrieb aus fünf handelsüblichen und leicht austauschbaren NiMh-Zellen der Größe AA
  - Bei Sonderbestückung auch Zellen in Größe AAA verwendbar
  - Integrierte Entladeschaltung zur Vermeidung des Memory-Effekts
  - Verpolungsschutz
  - Messung der Gesamtstromaufnahme und der Akkuspannung durch den Mikrocontroller
- Erweiterungsmöglichkeiten
  - Anschlussmöglichkeit an das SensOrchestra-System
  - Erweiterungsmöglichkeiten durch Steckverbinder
  - Komplette Dokumentation und frei zugängliche Schaltungsunterlagen ermöglichen eigene Projekte
  - Bluetooth zur seriellen Datenübertragung

2 Erste Schritte

Dieses Kapitel stellt unter den Kapiteln dieses Handbuchs insofern eine Ausnahme dar, als es den Charakter eines Tutorials aufweist. Es führt an Hand von Beispielen in die Programmierung des AVR-Mikrocontrollers auf der SandboxS 2 ein.

Zu Beginn wird auf die Einrichtung der Entwicklungsumgebung und der Hardware eingegangen. Danach soll ein möglichst einfaches Programm für den AVR-Mikrocontroller, das eine der LEDs auf der Sandbox steuert, erstellt werden. Ebenfalls wird das Erstellen eines neuen Projekts veranschaulicht. Das darauf folgende Beispiel soll einen ersten Einblick in die Steuerung mit dem Robotics Chips geben. Der Hexapod darf erst **nach** diesem Kapitels zusammengebaut werden. Da jeder einzelne Servomotor kalibriert werden muss.

2.1 Systemvoraussetzungen

- Handelsüblicher Computer oder Notebook mit USB Schnittstelle. Eine serielle/parallele Schnittstelle wird ebenfalls empfohlen.
- Microsoft Betriebssystem der Version *Windows 2000* oder *Windows XP* wird vorausgesetzt. Microsoft *Windows Vista* kann noch nicht verwendet werden, da die Unterstützung einzelner Programme fehlt. Die Entwicklung auf Unix/Linux Systemen ist ebenfalls möglich, wird in diesem Dokument aber nicht behandelt.
- Atmel Entwicklungsumgebung *AVR Studio* ab Version 4.13. Be ndet sich auf der beigelegten CD, im Ordner `AtmelAvrStudio`. Die aktuelle Version kann auch unter `http://www.atmel.com/avrstudio` bezogen werden.
- *WinAVR* Kompiler ab Version 20070525. Be ndet sich auf der beigelegten CD, im Ordner `WinAVR`. Die aktuelle Version kann auch unter `http://winavr.sourceforge.net` bezogen werden.
- Terminal Programm nach Wahl. In diesem Tutorial wird jedoch *HTerm* verwendet. Be ndet sich auf der beigelegten CD, im Ordner `HTerm`. Die aktuelle Version kann auch unter `http://www.der-hammer.info/terminal` bezogen werden.

Da Fehler in neueren Versionen der Programme nicht ausgeschlossen werden können, wird die Installation von der CD empfohlen. Diese Versionen wurden im Zusammenspiel miteinander getestet.

2.2 Entwicklungsumgebung einrichten

Um mit der Entwicklung der Firmware beginnen zu können, muss die Entwicklungsumgebung eingerichtet werden. Sie besteht aus den Programmen *WinAVR* und *AVRStudio*. Beide Programme be ndet sich auf der beigelegten CD. Wichtig ist, dass *WinAVR* vor dem *AVRStudio* installiert wird. Die Installation der Software wird im nachfolgenden Abschnitt ausführlich erklärt.

2.2.1 WinAVR installieren

Die Installation von WinAVR gestaltet sich sehr einfach. Man folgt den Anweisungen des Assistenten um die Installation abzuschließen. Eine bebilderte Installationsanleitung ist in Abbildung 2.1 dargestellt. Die Software be ndet sich im Ordner `WinAVR` auf der beigelegten CD.

WinAVR beinhaltet alle Programme um C und C++ (eingeschränkt) Anwendung auf der AVR Plattform zu entwickeln. Dazu zählen Kompiler, Assembler, Simulator, Debugger und Programmer. Weiters enthält es die *C Standard-Bibliothek für AVR (avr-libc)* und deren Dokumentation. Der Kompiler basiert auf der *GNU Compiler Collection (GCC)* und wurde speziell an die AVR Plattform angepasst. Das komplette Paket steht unter der *General Public License (GPL)* Lizenz und kann daher ohne zusätzliche Gebühren verwendet werden.

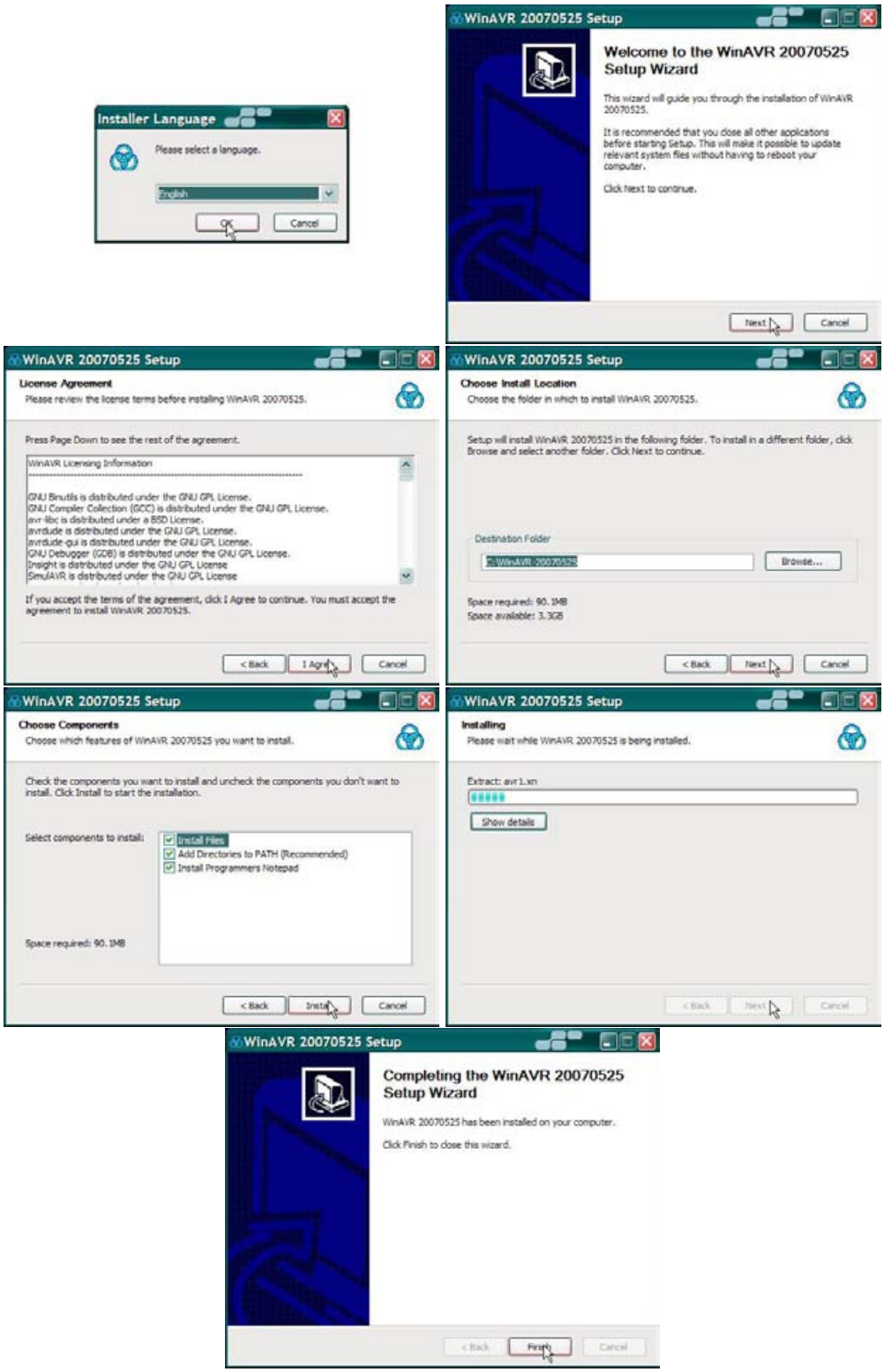

Installer Language
Please select a language.
English
OK
Cancel
WinAVR 20070525 Setup
Welcome to the WinAVR 20070525 Setup Wizard
This wizard will guide you through the installation of WinAVR 20070525.
It is recommended that you close all other applications before starting Setup. This will make it possible to update relevant system files without having to reboot your computer.
Click Next to continue.
Next
Cancel
WinAVR 20070525 Setup
License Agreement
Please review the license terms before installing WinAVR 20070525.
Press Page Down to see the rest of the agreement.
WinAVR Licensing Information
GNU Binutils is distributed under the GNU GPL License.
GNU Compiler Collection (GCC) is distributed under the GNU GPL License.
avr-libc is distributed under a BSD License.
avrdude is distributed under the GNU GPL License.
avrdude-gui is distributed under the GNU GPL License.
GNU Debugger (GDB) is distributed under the GNU GPL License.
Insight is distributed under the GNU GPL License
SimulAVR is distributed under the GNU GPL License
If you accept the terms of the agreement, click I Agree to continue. You must accept the agreement to install WinAVR 20070525.
< Back
I Agree
Cancel
WinAVR 20070525 Setup
Choose Install Location
Choose the folder in which to install WinAVR 20070525.
Setup will install WinAVR 20070525 in the following folder. To install in a different folder, click Browse and select another folder. Click Next to continue.
Destination Folder
C:\WinAVR-20070525
Browse...
Space required: 90.1MB
Space available: 3.3GB
< Back
Next
Cancel
WinAVR 20070525 Setup
Choose Components
Choose which features of WinAVR 20070525 you want to install.
Check the components you want to install and uncheck the components you don't want to install. Click Install to start the installation.
Select components to install:
Install Files
Add Directories to PATH (Recommended)
Install Programmers Notepad
Space required: 90.1MB
< Back
Install
Cancel
WinAVR 20070525 Setup
Installing
Please wait while WinAVR 20070525 is being installed.
Extract: avr1.xn
Show details
< Back
Next
Cancel
WinAVR 20070525 Setup
Completing the WinAVR 20070525 Setup Wizard
WinAVR 20070525 has been installed on your computer.
Click Finish to close this wizard.
< Back
Finish
Cancel

Abbildung 2.1: Schrittweises Installieren des WinAVR Kompilers.

2.2.2 Atmel AVR Studio installieren

Die Entwicklungsumgebung be ndet sich ebenfalls auf der Begleit-CD im Ordner `AtmelAvrStudio`. Die Installation darf erst **nach** erfolgreicher Installation des WinAVR Pakets erfolgen. Sie wird durch einen Assistenten begleitet. Wichtig ist dabei, dass der *Jungo* USB-Treiber auch ausgewählt wird. Eine bebilderte Installationsanleitung ist in Abbildung 2.2 dargestellt.

Das AVR Studio beinhaltet eine komfortable graphische Benutzeroberfläche sowie Assembler, Debugger und Simulator. Um Anwendungen in C/C++ entwickeln zu können, benötigt es das zuvor installierte WinAVR Paket. Dieses Paket wird in das AVR Studio integriert und ermöglicht eine komfortable Programmentwicklung. Auf die Kommandozeilenprogramme des WinAVR Pakets muss daher nicht zurückgegriffen werden.

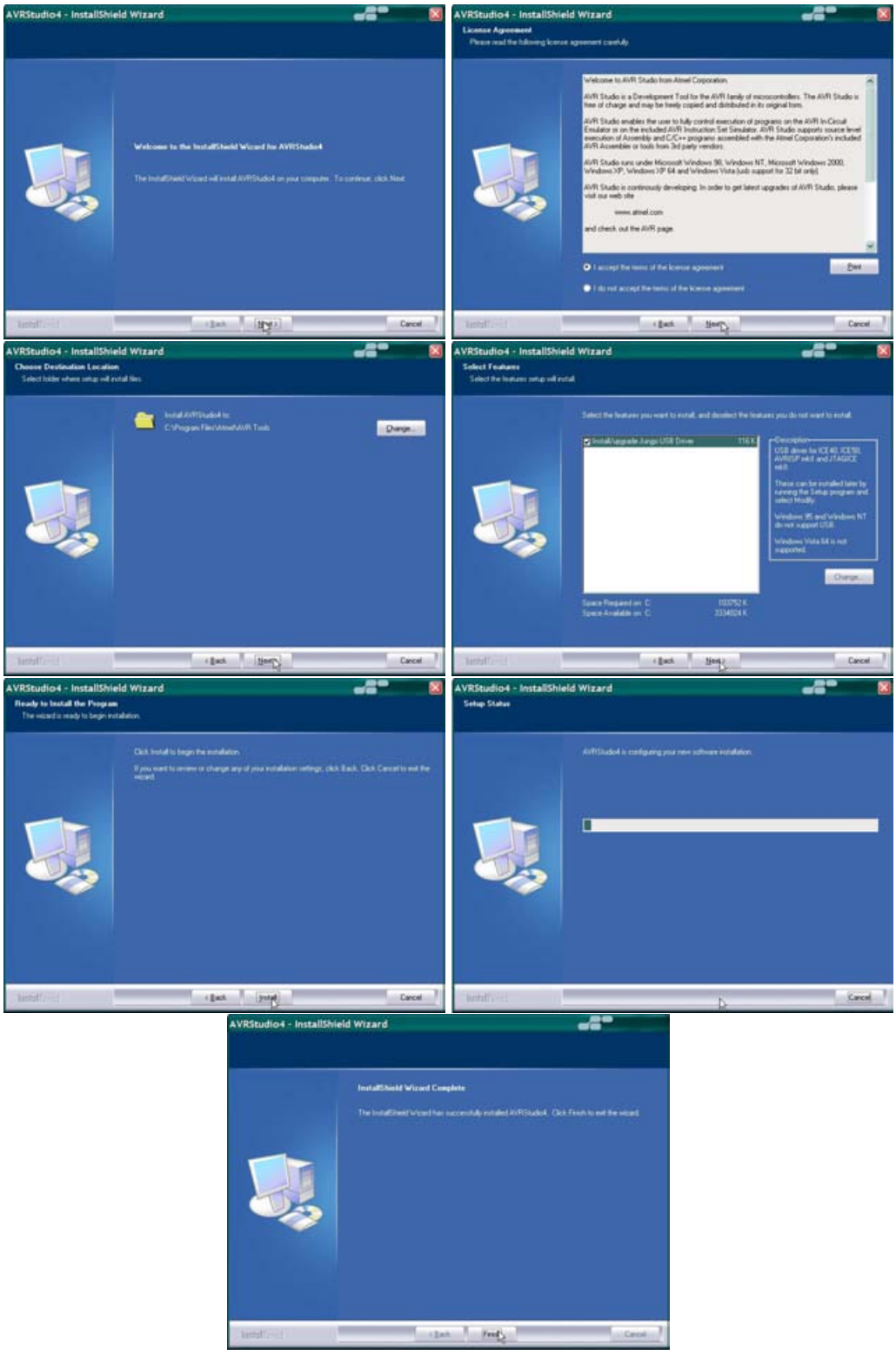


Abbildung 2.2: Installation der Atmel AVR Entwicklungsumgebung.

2.3 Gerätetreiber installieren

Da beide Geräte, der *JTAG ICE* und die SandboxS 2, über das USB Bussystem mit dem Computer kommunizieren, ist für jedes ein Gerätetreiber erforderlich. Der nötige Treiber für das JTAG ICE wurde bereits mit dem AVR Studio installiert. Der SandboxS 2 Treiber be ndet sich auf der CD im Ordner `Driver`.

2.3.1 JTAG ICE Mk II Treiber installieren

Das JTAG ICE Mk II ist ein Gerät zur Fehlersuche (debugging) von Programmen, welche bereits auf der Hardware (SandboxS 2) laufen. Es verwendet den *Jungo* USB-Treiber des AVR Studios.

Beim erstmaligen Anstecken des JTAGs an die USB Schnittstelle, ndet das Betriebssystem ein neues Gerät. Der JTAG ICE muss natürlich bereits eingeschaltet sein. Um eine reibungslose Treiberinstallation zu gewährleisten, müssen alle Aktionen wie in Abbildung 2.3 ausgeführt werden.

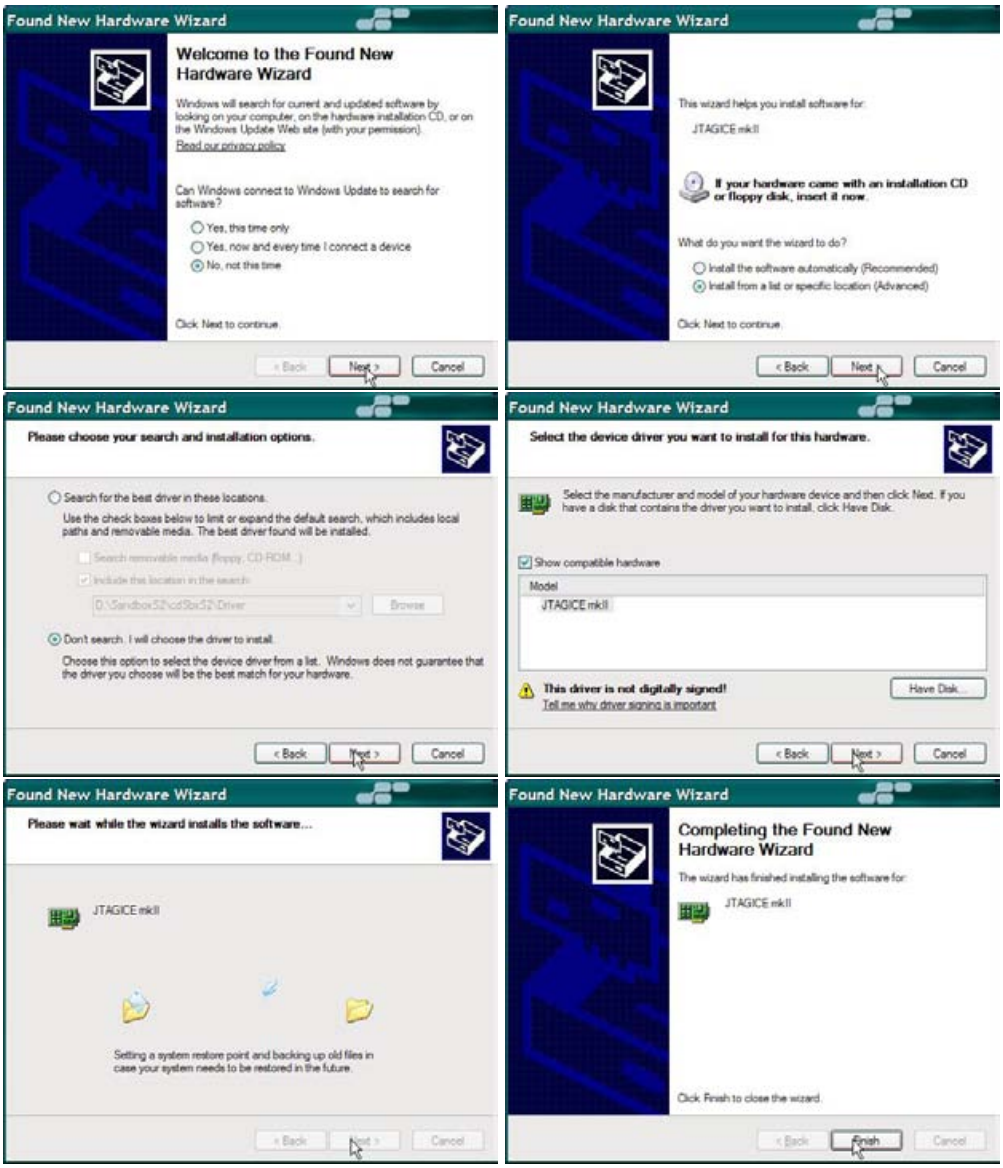

Found New Hardware Wizard
Welcome to the Found New Hardware Wizard
Windows will search for current and updated software by looking on your computer, on the hardware installation CD, or on the Windows Update Web site (with your permission).
Read our privacy policy
Can Windows connect to Windows Update to search for software?
Yes, this time only
Yes, now and every time I connect a device
No, not this time
Click Next to continue.
< Back
Next >
Cancel
Found New Hardware Wizard
This wizard helps you install software for:
JTAGICE mkII
If your hardware came with an installation CD or floppy disk, insert it now.
What do you want the wizard to do?
Install the software automatically (Recommended)
Install from a list or specific location (Advanced)
Click Next to continue.
< Back
Next >
Cancel
Found New Hardware Wizard
Please choose your search and installation options.
Search for the best driver in these locations.
Use the check boxes below to limit or expand the default search, which includes local paths and removable media. The best driver found will be installed.
Search removable media (floppy, CD-ROM...)
Include this location in the search:
D:\Sandbox32\cdSbx32\Driver
Browse
Don't search. I will choose the driver to install.
Choose this option to select the device driver from a list. Windows does not guarantee that the driver you choose will be the best match for your hardware.
< Back
Next >
Cancel
Found New Hardware Wizard
Select the device driver you want to install for this hardware.
Select the manufacturer and model of your hardware device and then click Next. If you have a disk that contains the driver you want to install, click Have Disk.
Show compatible hardware
Model
JTAGICE mkII
This driver is not digitally signed!
Tell me why driver signing is important
Have Disk...
< Back
Next >
Cancel
Found New Hardware Wizard
Please wait while the wizard installs the software...
JTAGICE mkII
Setting a system restore point and backing up old files in case your system needs to be restored in the future.
< Back
Next >
Cancel
Found New Hardware Wizard
Completing the Found New Hardware Wizard
The wizard has finished installing the software for:
JTAGICE mkII
Click Finish to close the wizard.
< Back
Finish
Cancel

Abbildung 2.3: Installieren des Atmel JTAG ICE Debuggers nach dem Anstecken an die USB Schnittstelle.

2.3.2 SandboxS 2 Treiber installieren

Der auf der SandboxS 2 verbaute Mikrocontroller besitzt bereits eine USB Unterstützung. Um eine einfache Kommunikation mit dem Computer zu ermöglichen, meldet sie sich als serielle Schnittstelle beim Betriebssystem an.

Bei der Installation wird ebenfalls ein spezieller Gerätetreiber benötigt. Das Betriebssystem erkennt nach dem Einschalten ein neues Gerät, und verlangt nach einem Treiber. Dieser be ndet sich auf der beigelegten CD im Ordner `Driver`. Die SandboxS 2 sollte zuvor mit voll geladenen Akkuzellen bestückt werden. (Abbildung 2.4). Um eine reibungslose Treiberinstallation zu gewährleisten müssen alle Aktionen wie in Abbildungen 2.5, 2.6 ausgeführt werden.

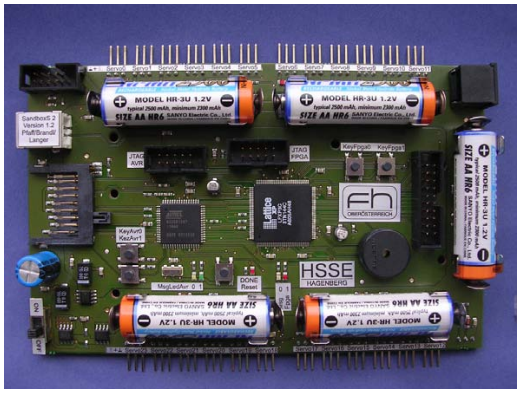


Abbildung 2.4: SandboxS2 mit bestückten Akkuzellen.

Die SandboxS 2 wird mit einer Firmware, die die USB Unterstützung des Mikrocontrollers aktiviert, ausgeliefert . Wird sie trotz mehrere Versuche nicht erkannt, könnte es an einer fehlenden/fehlerhaften Firmware liegen. Sollte dies der Fall sein, kann die Installation erst nach dem Erstellen des ersten Beispielprogramms fortgesetzt werden.

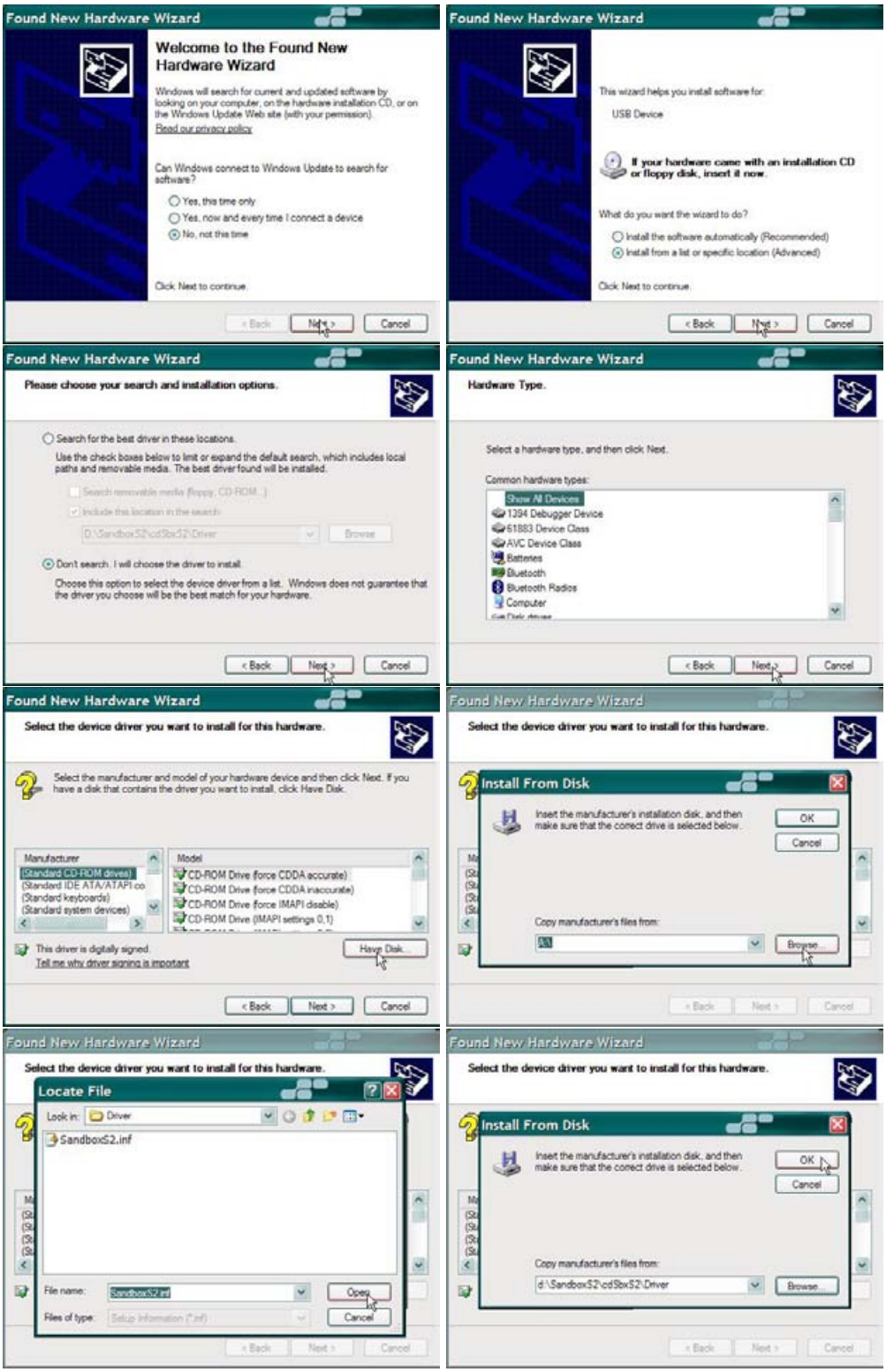

Found New Hardware Wizard
Welcome to the Found New Hardware Wizard
Windows will search for current and updated software by looking on your computer, on the hardware installation CD, or on the Windows Update Web site (with your permission).
Read our privacy policy
Can Windows connect to Windows Update to search for software?
Yes, this time only
Yes, now and every time I connect a device
No, not this time
Click Next to continue.
< Back
Next >
Cancel
Found New Hardware Wizard
This wizard helps you install software for:
USB Device
If your hardware came with an installation CD or floppy disk, insert it now.
What do you want the wizard to do?
Install the software automatically (Recommended)
Install from a list or specific location (Advanced)
Click Next to continue.
Found New Hardware Wizard
Please choose your search and installation options.
Search for the best driver in these locations.
Use the check boxes below to limit or expand the default search, which includes local paths and removable media. The best driver found will be installed.
Don't search. I will choose the driver to install.
Choose this option to select the device driver from a list. Windows does not guarantee that the driver you choose will be the best match for your hardware.
Found New Hardware Wizard
Hardware Type.
Select a hardware type, and then click Next.
Common hardware types:
Show All Devices
1394 Debugger Device
61883 Device Class
AVC Device Class
Batteries
Bluetooth
Bluetooth Radios
Computer
Found New Hardware Wizard
Select the device driver you want to install for this hardware.
Select the manufacturer and model of your hardware device and then click Next. If you have a disk that contains the driver you want to install, click Have Disk.
Manufacturer
(Standard CD-ROM drives)
(Standard IDE ATA/ATAPI co
(Standard keyboards)
(Standard system devices)
Model
CD-ROM Drive (force CDDA accurate)
CD-ROM Drive (force CDDA inaccurate)
CD-ROM Drive (force IMAPI disable)
CD-ROM Drive (IMAPI settings 0,1)
This driver is digitally signed.
Tell me why driver signing is important
Have Disk...
Install From Disk
Insert the manufacturer's installation disk, and then make sure that the correct drive is selected below.
OK
Cancel
Copy manufacturer's files from:
Browse...
Found New Hardware Wizard
Select the device driver you want to install for this hardware.
Locate File
Look in:
Driver
SandboxS2.inf
File name:
Open
Files of type:
Cancel
Found New Hardware Wizard
Select the device driver you want to install for this hardware.
Install From Disk
Insert the manufacturer's installation disk, and then make sure that the correct drive is selected below.
OK
Cancel
Copy manufacturer's files from:
d:\SandboxS2\cdSbxS2\Driver
Browse...

Abbildung 2.5: Installieren des SandboxS 2 Treibers nach dem Anstecken an die USB Schnittstelle (Teil 1).

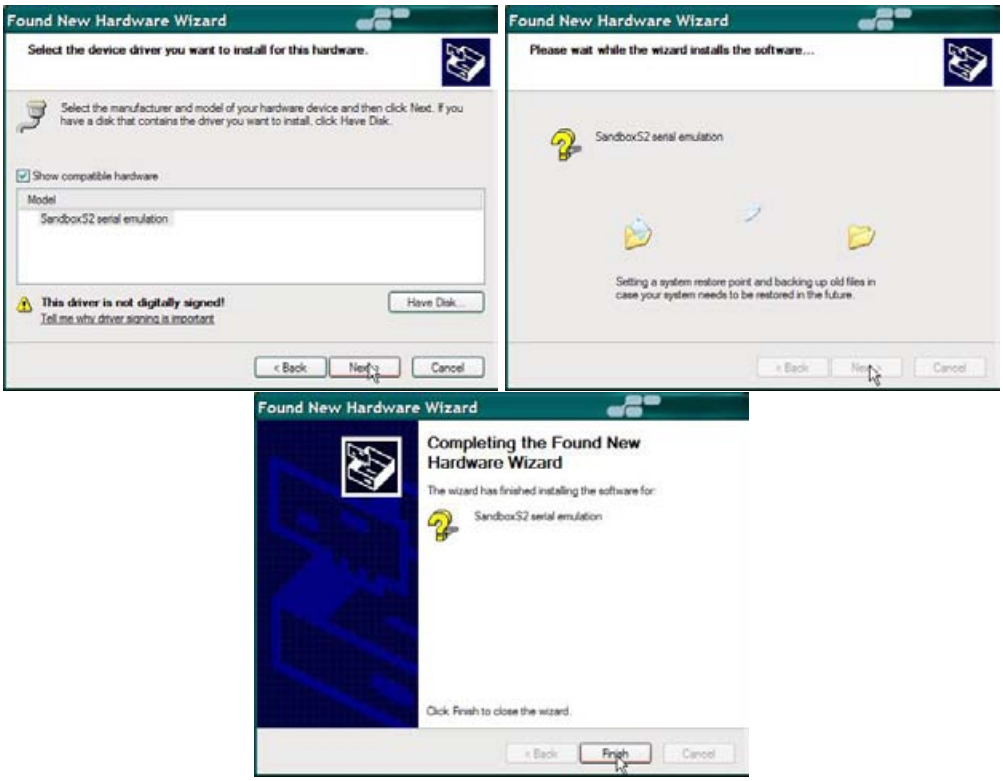

Found New Hardware Wizard
Select the device driver you want to install for this hardware.
Select the manufacturer and model of your hardware device and then click Next. If you have a disk that contains the driver you want to install, click Have Disk.
Show compatible hardware
Model
SandboxS2 serial emulation
This driver is not digitally signed!
Tell me why driver signing is important
Have Disk...
< Back
Next >
Cancel
Found New Hardware Wizard
Please wait while the wizard installs the software...
SandboxS2 serial emulation
Setting a system restore point and backing up old files in case your system needs to be restored in the future.
< Back
Next >
Cancel
Found New Hardware Wizard
Completing the Found New Hardware Wizard
The wizard has finished installing the software for:
SandboxS2 serial emulation
Click Finish to close the wizard.
< Back
Finish
Cancel

Abbildung 2.6: Installieren des SandboxS 2 Treibers nach dem Anstecken an die USB Schnittstelle (Teil 2).

2.4 „Hello World“ in der Sandbox-Version

Da nun die benötigte Hard- und Software installiert wurde, kann mit einem ersten Programm begonnen werden. Dazu werden folgende Punkte beschrieben:

- Projekt erstellen
- Programm schreiben
- Projekt übersetzen
- Programm ausführen
- Programm testen
- Zusammenfassung
- Aufgaben

2.4.1 Projekt erstellen

Ein Projekt im AVR Studio beinhaltet verschiedenste Einstellungen. Darunter Einstellungen für Mikrocontroller, Debugger, Frequenz, Projektverzeichnisse und verwendete Dateien. Um ein neuen Projekt zu erstellen gibt es zwei unterschiedliche Möglichkeiten. Entweder verwendet man ein bereits vorhandenes Projekt und kopieren dieses, oder man erstellt ein neues leeres Projekt und passt alle Einstellungen an. Beide Lösungen werden nachfolgend gezeigt.

2.4.1.1 Projekt mit Vorlage erstellen

Der einfachste Weg ein neues Projekt zu beginnen, ist eine bereits fertig angepasste Projektdatei zu kopieren. Die hierfür benötigten Dateien be nden sich auf der CD im Ordner `HexapodFirmware`. Es müssen die Ordner `Template` und `lib` in ein lokales Verzeichnis kopiert werden. Beide Ordner müssen in der selben Verzeichnishierarchie verbleiben. Der Ordner `lib` enthält die Dateien der Softwarebibliothek, mit denen Programme für den Hexapod entwickelt werden können. Der Ordner „Template“ enthält eine fertig kon gurierte Projektdatei (`Hexapod.aps`) und ein einfaches Anwendungsskelett (`main.c`). Durch einen Doppelklick auf die Projektdatei wird das Projekt geöffnet.

2.4.1.2 Projekt ohne Vorlage erstellen

Es werden nun alle nötigen Schritte erklärt um ein komplett neues Projekt einzurichten.
Nach dem Start des AVR Studios sollte ein Dialog mit dem Titel „Welcome to AVR Studio 4“ angezeigt werden. Ist dieser Dialog nicht zu sehen, kann er über den Menübefehl „New Project->Project Wizard“ aufgerufen werden. Ein Klick auf die Schaltfläche „New Project“wechselt auf die nächste Seite des Assistenten.

• Projektbezeichnung und Pfad
Unter „Project type“ ist „AVR GCC“ auszuwählen. Der Pfad für das neue Projekt ist ebenfalls zu setzen. Weiters sollte dem Projekt ein sprechender Projektname verliehen werden. Durch einen Klick auf die Schaltfläche „Next »“ werden die Eingaben übernommen und der Assistent wechselt auf die

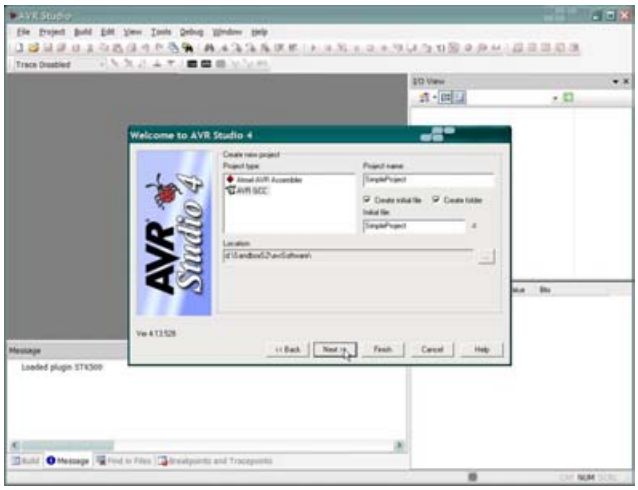


Abbildung 2.7: Projekt Assistent: Eingabe des Projektnamens.

nächste Seite (Abbildung 2.7).

• Debugger und Mikrocontroller

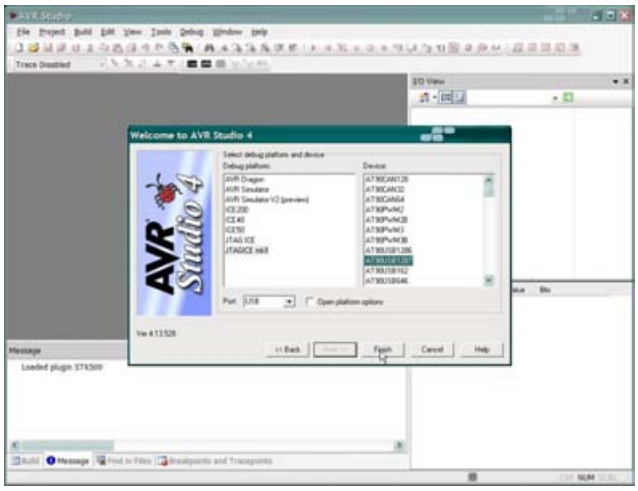


Abbildung 2.8: Projekt Assistent: Einstellen der verwendeten Hardware.

Auf dieser Seite werden nun der verwendete Debugger und der Mikrocontroller eingestellt. Verwendet wird das „JTAG ICE Mk II“ und der „AT90USB1287“ Mikrocontroller. Durch einen Klick auf die Schaltfläche „Finish“ wird der Assistent beendet (Abbildung 2.8).

• Hexapod Softwarebibliothek

Die Projektdateien wurden nun auf der Festplatte erzeugt. Als nächstes muss die Hexapod Softwarebibliothek von der CD auf den Computer kopiert werden. Dazu wird der Ordner `lib` im `HexapodFirmware` Verzeichnis auf die Festplatte kopiert. Dieser beinhaltet den gesamten Quell-

code der Bibliothek.

- **Dateien hinzufügen**

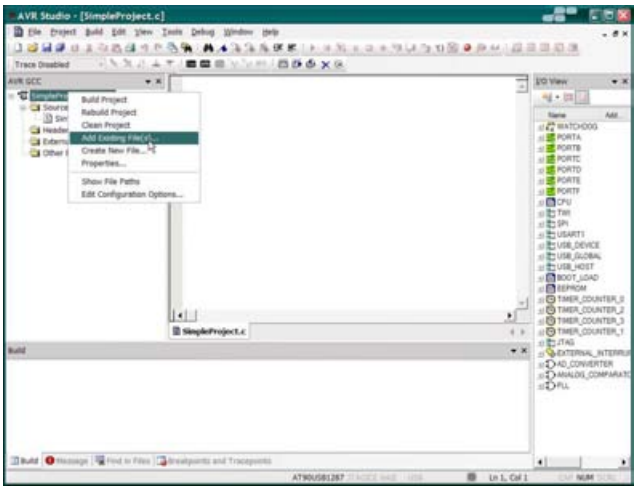


Abbildung 2.9: Hinzufügen bereits existierender Dateien.

Im nächsten Schritt müssen alle Dateien der verwendeten Hexapod Software-Bibliotheken hinzugefügt werden. Hierzu, Rechtsklick auf die Projektwurzel im linken Fenster und Befehl „Add existing File(s)...“ auswählen (Abbildung 2.9). Im nun erscheinenden Dialog werden alle benötigten Dateien ausgewählt. Dieser Vorgang muss solange durchgeführt werden, bis alle Dateien der folgenden Ordner und Unterordner im Projekt eingebunden sind:

```
lib/SandboxS2Lib/
lib/uart/
lib/uart_usb/
lib/RoboticsChip/
lib/InverseKinematics/
lib/MemoryCard/
```

- **Grundeinstellungen**

Danach sind die Projekt-Einstellungen anzupassen. Der benötigte Dialog be ndet sich unter dem Menüpunkt „Project->Con guration Options“. Hier wird unter „Frequency“ der Takt des Mikrocontrollers auf „8000000“ (8MHz) eingestellt. Mit der Einstellung „Optimization“ lässt sich die Optimierung des Kompilers einschalten. Diese Option sollte jedoch nur bei fertigen Programmen, vor der Auslieferung, eingestellt werden. Danach wird auf die Karte „Include Directories“ gewechselt (Abbildung 2.10).

- **Hexapod Softwarebibliothek**

Nun muss dem Kompiler noch mitgeteilt werden, in welchem Ordner er nach den Header-Dateien (Dateierweiterung `.h`) der Hexapod Softwarebibliothek suchen soll. Hierzu werden alle Pfade der einzelnen Ordner der Bibliothek in die Liste eingetragen (Abbildung 2.11). Der Pfad ist abhängig von der Position der Hexapod Bibliothek auf der Festplatte. Nachfolgend ein Beispiel mit relativer Verzeichnisangabe. Bei diesem be ndet sich der `lib` Ordner im selben Verzeichnis wie die Projketdatei

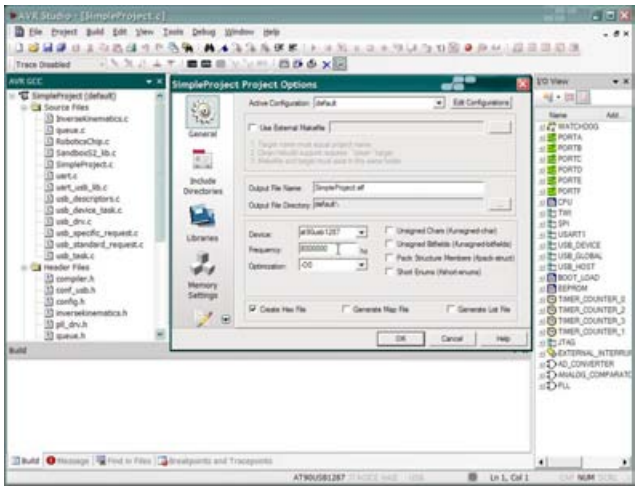


Abbildung 2.10: Projekt Grundeinstellungen.

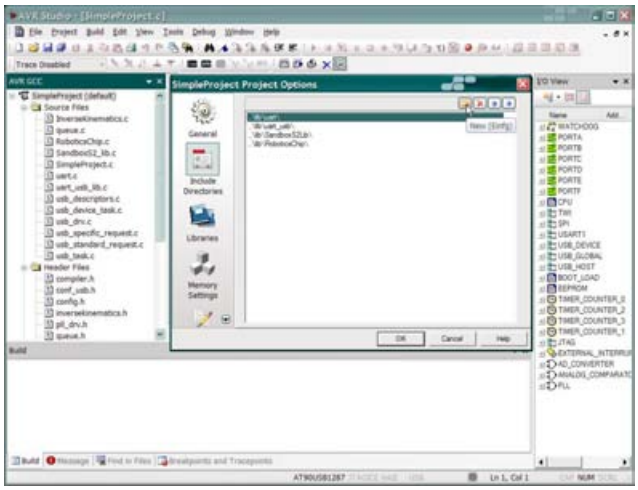


Abbildung 2.11: Hexapod Softwarebibliothek Header-Dateien hinzufügen.

(Dateierweiterung `.aps`).

```
./lib/SandboxS2Lib/
./lib/uart/
./lib/uart_usb/
./lib/RoboticsChip/
./lib/InverseKinematics/
./lib/MemoryCard/
```

Danach wird auf die Karte „Custom Options“ gewechselt.

• Kompiler Einstellungen

Zum Abschluss wird der WinAVR Kompiler ausgewählt. Hierzu muss der Pfad zu den zwei Programmen stimmen (avr-gcc und make). Dies müssen jedoch nur verändert werden, wenn das AVR Studio den Pfad falsch ermittelt hat, oder eine andere WinAVR Version verwendet wird (Abbildung 2.12).

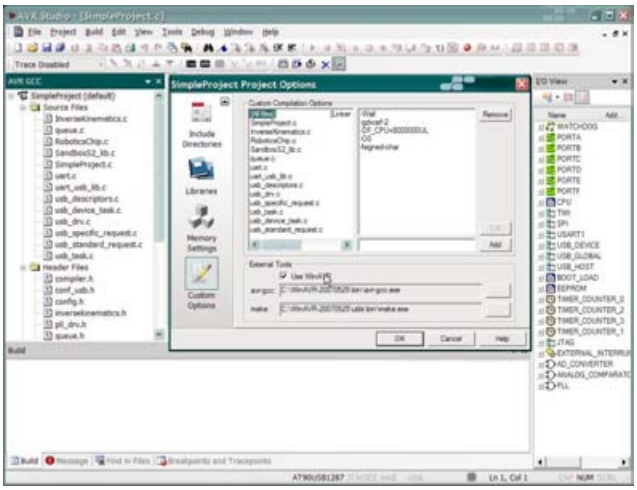


Abbildung 2.12: Kompiler Einstellungen.

2.4.2 Programm schreiben

Ein neues Projekt sollte nun bereit sein. Auf welche Weise es erstellt wurde ist für dieses Beispiel unerheblich.

2.4.2.1 Programmspezifikation

Das Programm bringt die LED Msg-Avr0 zum Aufleuchten. Wenn man die Taste Key_Avr0 drückt leuchtet auch die LED Msg-Avr1 auf. Zusätzlich wird die Zeichenkette „Hello World!“ über die USB und UART Schnittstellen ausgegeben.

2.4.2.2 Quellcode

```
/*************************************************************************
*  Workfile    : main.c
*  Author      : Christian Raschko
*  Date        : 2007-08-20
*  Description : main program
*  Revision    : 3
*************************************************************************/


//////////////////////////////////////////////////////////////////////////
// includes
//////////////////////////////////////////////////////////////////////////

//standard library includes
#include <stdio.h>
//AVR specific includes
#include <avr/interrupt.h>
//hexapod library includes
#include "SandboxS2_lib.h"
#include "uart.h"
#include "uart_usb_lib.h"


//////////////////////////////////////////////////////////////////////////
// global variables/definitions
//////////////////////////////////////////////////////////////////////////
FILE * gStrmUSB  = NULL;
FILE * gStrmUART = NULL;


//////////////////////////////////////////////////////////////////////////
// functions
//////////////////////////////////////////////////////////////////////////

//stream I/O wrapper functions
int strm_uart_putc(char c,FILE *s) {uart_putc(c);return 0;}
int strm_uart_getc(FILE *s)        {return (unsigned char)uart_getc();}
int strm_usb_putc(char c,FILE *s)  {uart_usb_putchar(c);return 0;}
int strm_usb_getc(FILE *s)         {return (unsigned char)uart_usb_getchar();}
```

```
//do all the initialization stuff here
void Init(void)
{
  //initialize sandboxs2 AVR peripherals
  SL2_Init();

  //initialize usb
  uart_usb_init();

  //initialize UART, set baud rate to 38400
  uart_init(UART_CALC_BAUDRATE(38400));

  //activate stream I/O
  gStrmUSB = fdevopen(strm_usb_putc,strm_usb_getc);
  gStrmUART = fdevopen(strm_uart_putc,strm_uart_getc);

  //enable interrupts
  sei();
}

//program entry point
int main (void)
{
  //Initialize system
  Init();

  //wait on key
  SL2_AvrSetMsgLed(SL2_MSG_LED_0, SL2_MSG_LED_STATUS_ON);
  SL2_AvrWaitOnKey(SL2_KEY_0, SL2_KEY_PRESSED);
  SL2_AvrSetMsgLed(SL2_MSG_LED_1, SL2_MSG_LED_STATUS_ON);

  //print welcome message
  fprintf(gStrmUSB, "Hello␣World!\n");
  fprintf(gStrmUART, "Hello␣World!\n");

  //never return, loop forever
  for(;;);
  return 0;
}
```

2.4.2.3 Bemerkungen

Die gesamte Hexapod Softwarebibliothek ist in einzelne Module aufgeteilt. Jedes Modul ist einer bestimmten Aufgabe zugeordnet. Alle Dateien eines Moduls be nden sich in dessen Ordner (z.B: `lib/uart/`). Um die Funktionen eines Moduls zu verwenden, bedarf es drei Schritten: Erstens, inkludieren der Header-Datei (z.B: `#include "uart.h"`), damit die Funktionen benützt werden können. Zweitens, initialisieren des Moduls (z.B: `uart_init`), dazu muss die jeweilige Initialisierungsfunktion aufgerufen werden. Im dritten Schritt muss dem Kompiler noch mitgeteilt werden, dass das Modul im aktuellen Projekt verwendet wird. Hierzu werden dem Projekt alle Dateien des Moduls hinzugefügt. Um Funktionen einzelnen Modulen zuordnen zu können, besitzt jede Funktionen einen eindeutigen Prä x (z.B: `uart_`). Die in diesem Beispiel verwendeten Module sind folgende:

Modul	Prä x
uart	uart_
uart_usb	uart_usb_
SandboxS2Lib	SL2_

Da einige Module *Interrupts* verwenden müssen diese aktiviert werden.

Die beiden Variablen gStrmUSB und gStrmUART zeigen auf Strukturen die jeweils ein Ein/Ausgabe-Gerät repräsentieren. Dadurch können mit den selben Funktionen unterschiedliche Datenströme gelesen oder geschrieben werden. Weitere Informationen sind in Abschnitt 9.5 zu nden.

Da die SandboxS 2 kein Betriebssystem verwendet, darf das Programm niemals zu Ende sein. Das Verhalten bei Verlassen des Programms ist somit nicht de niert.

Die Funktion fprintf() sollte bereits bekannt sein. Sie entspringt der C Standard Bibliothek. Eine AVR spezi sche Anpassung stellt jedoch die Funktion fdevopen() dar, sie soll die fopen() Funktion am AVR ersetzen.

2.4.3 Projekt übersetzen

Um das Projekt zu übersetzen wird der Menüpunkt „Build->Build“ ausgewählt. Nun generiert das AVR Studio ein so genanntes *Makefile* und lässt es von dem Programm *GNU Make*, das Teil des WinAVR Pakets ist, ausführen. Dieses Programm ruft in richtiger Reihenfolge, für jede Datei, den Kompiler auf. Die Ausgabe kann man im Fenster „Build“ am unteren Bildschirmrand sehen (Abbildung 2.13). Sind Fehler (Errors) aufgetreten, werden diese mit einem roten Punkt in der Liste gekennzeichnet. Zusätzlich wird die Anzahl der Fehler am Ende der Ausgabe angezeigt. Das gleiche gilt für Warnungen (Warnings), nur ist der Punkt diesmal gelb. Nachfolgende Warnungen können ignoriert werden:

```
delay.h:89:3: warning: #warning "Compiler optimizations disabled;
functions from <util/delay.h> won't work as designed"
```

Diese Warnung tritt nur auf wenn die Optimierung des Kompilers abgeschaltet ist. Die Wartefunktionen (z.B: _delay_ms()) sind dadurch etwas ungenauer. Die Warnung lässt sich durch einschalten der Optimierung beheben.

```
usb_standard_request.c:335:6: warning: #warning with avrgcc
assumes devices descriptors are stored in the lower 64Kbytes of
on-chip flash memory
```

Diese Warnung kann gänzlich ignoriert werden. Beachten muss man sie nur, wenn man Konstanten verwendet die größer als 64Kbytes sind (z.B: Bilder, Audio). Dabei kann es sein, dass die USB-Deskriptor Tabelle ausserhalb der 64Kbyte Grenze (im Flash-Speicher) gelegt wird (vom Kompiler) und nicht mehr gelesen werden kann. Zur Lösung des Problems wird auf die Datei conf_usb.h Zeile 149 verwiesen.

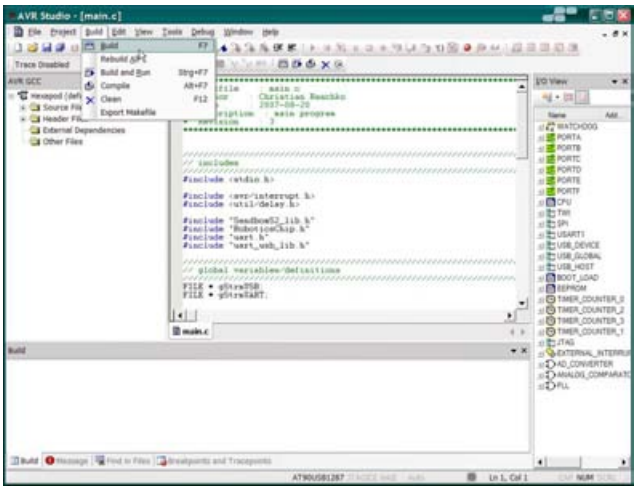


Abbildung 2.13: Projekt übersetzen.

2.4.4 Programm ausführen

Um das nun erfolgreich übersetzte Programm auf der SandboxS 2 auszuführen, gibt es zwei Möglichkeiten. Das Programm wird in beiden Fällen auf den Mikrocontroller übertragen. Da der Mikrocontroller über einen Flash-Speicher verfügt bleibt das Programm auch nach dem Ausschalten erhalten. Bei der ersten Variante wir mit dem Programmier-Dialog gearbeitet, bei der Zweiten mit dem Debugger.
Bevor mit der Übertragung begonnen werden kann, muss die Sandbox mit dem JTAG ICE verbunden werden (Abbildung 2.14).

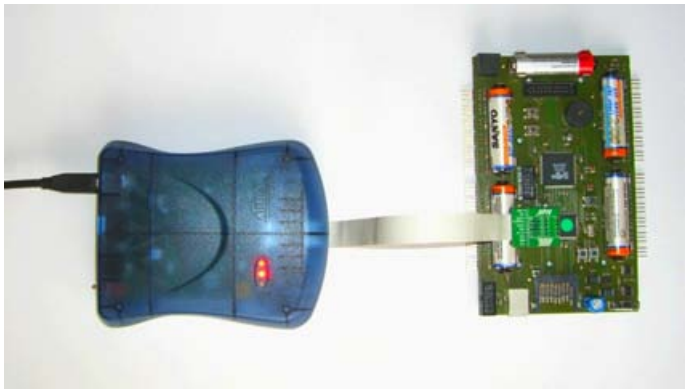


Abbildung 2.14: SandboxS2 mit angeschlossenem JTAG ICE.

2.4.4.1 Verwenden des Programmier-Dialogs

Über diesen Dialog lassen sich sämtliche Einstellungen des Mikrocontroller verändern. Dadurch kann er aber auch irreparabel beschädigt werden. Keinesfalls dürfen die Einstellungen in den Registerkarten „Fuses“ oder „LockBits“ willkürlich verändert werden. Die erste Registerkarte beinhaltet alle Funktionen die zur Übertragung benötigt werden.

- **Auswahl des Programmiergerätes**

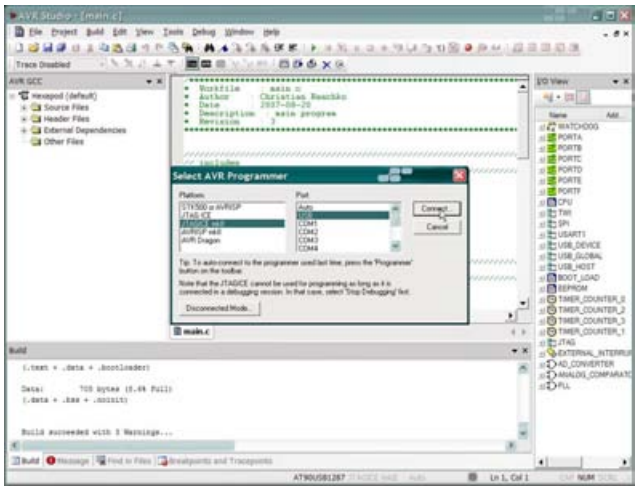


Abbildung 2.15: Auswahl des Programmiergerätes.

Im Menu wird der Punkt „Tools->Program AVR->Connect...“ ausgewählt. Ein Dialog erscheint, in dem man das verwendete Programmiergerät auswählen kann. Hier wird „JTAG ICE Mk II“ und „USB“ ausgewählt (Abbildung 2.15).

- **Übertragung des Programms**

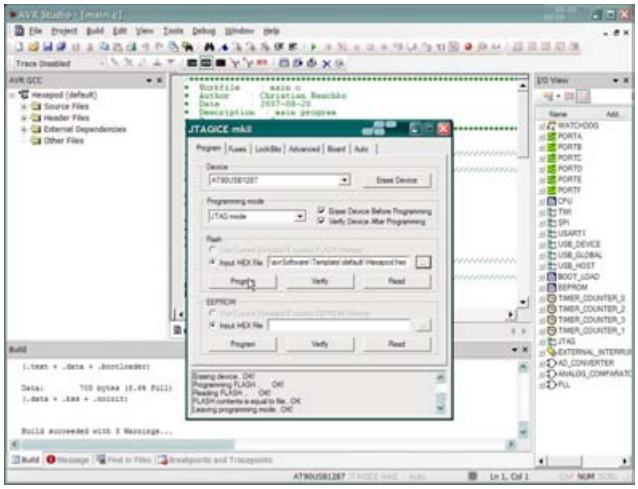


Abbildung 2.16: Übertragung des übersetzen Programms auf den Mikrocontroller.

Ist ein Gerät an dem gewählten Anschluss vorhanden, wird ein neuer Dialog angezeigt. In diesem wird der Mikrocontroller auf „AT90USB1287“ gestellt und der Modus auf „JTAG mode“. Nun kann man die zu übertragende Datei auswählen (Dateierweiterung `.hex`). Sie be ndet sich im Projektverzeichnis, im Unterordner `default`. Mit der Schaltfläche „Program“ wird die Datei in den Mikrocontroller übertragen (Abbildung 2.16). Sobald die Übertragung abgeschlossen wurde, wird das

Programm ausgeführt. Durch einen *Reset*, der mit der „Reset“ Taste ausgelöst werden kann, fängt das Programm wieder vom Anfang an.

2.4.4.2 Verwenden des Debuggers

Mit dem Debugger können Programme Schritt für Schritt, oder bis zu bestimmten Haltepunkten (Breakpoints) ausgeführt werden. Weiters ist es möglich Werte von Variablen oder Speicherbereichen während der Abarbeitung anzeigen zu lassen. Ein mächtiges Werkzeug zur Fehlersuche. Jedoch gibt es Einschränkung bei der Verwendung der USB-Funktion. Wird der Prozessor durch den Debugger angehalten, können die USB Nachrichten vom Computer nicht mehr durch den Mikrocontroller quittiert werden. Nach kurzer Zeit zeigt das Betriebssystem ein fehlerhaftes Gerät an . Zusätzlich können sich die *Interrupt Service Routinen (ISR)* störend auswirken.

- **Debugger starten**

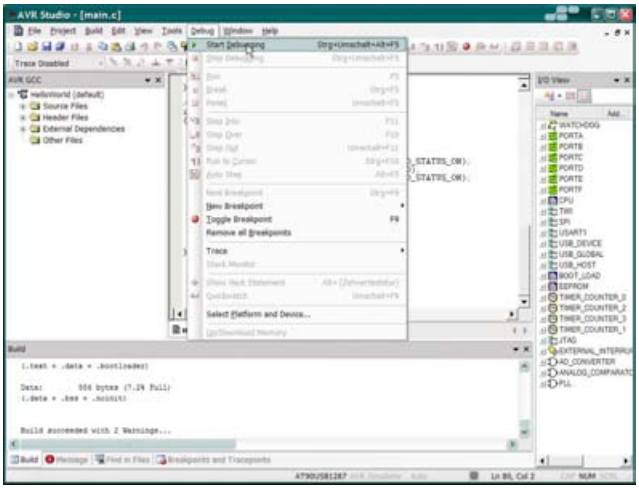


Abbildung 2.17: Starten des Debuggers.

Im Menu wird der Punkt „Debug->Start Debugging“ ausgewählt. Der Debugger versucht nun eine Verbindung mit der Sandbox aufzubauen. Ist die Sandbox oder der JTAG ICE nicht angeschlossen/eingeschaltet erscheint eine Fehlermeldung. Ist eine Verbindung vorhanden wird das Programm übertragen, dies kann an dem Fortschrittsbalken in der Statusleiste erkannt werden (Abbildung 2.17).

- **Debugger ist bereit**

Hat der Debugger das Programm erfolgreich übertragen, ist er für die Annahme weiterer Befehle bereit. Der gelbe Zeiger am rechten Rand des Editors zeigt immer auf den als nächstes abzuarbeitenden Befehl. Am Anfang steht er knapp unterhalb der `main` Funktion (Abbildung 2.18).

- **Steuern des Debuggers**

Der Debugger besitzt Befehle die den Programmablauf beeinflussen können. Diese werden nachfolgend erklärt (Abbildung 2.19):

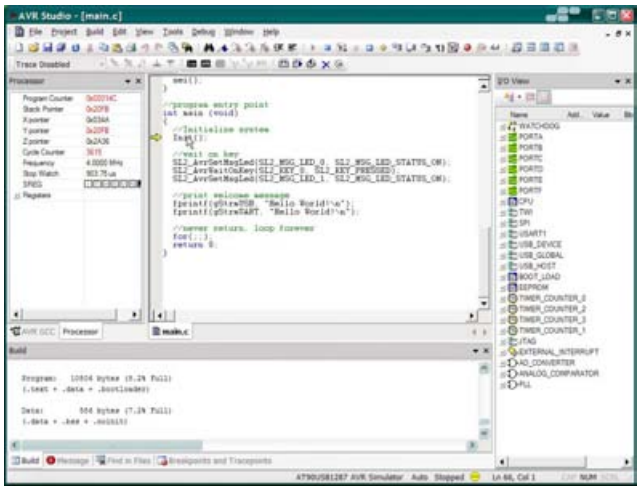


Abbildung 2.18: Debugger ist bereit zum Ausführen des Programms.

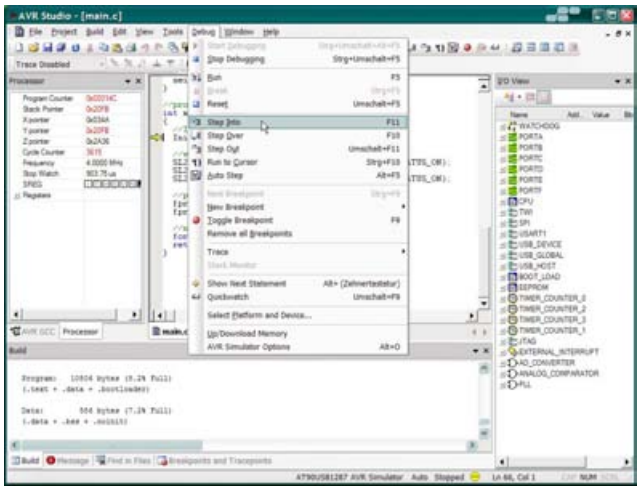


Abbildung 2.19: Steuerbefehle des Debuggers.

Stop	Beendet die debugging Sitzung.
Run	Führt das aktuelle Programm aus.
Break	Pausiert die Ausführung ab.
Reset	Setzt das Programm auf den Anfangszustand zurück.
Step Into	Führt einen einzigen Befehl aus, springt dabei in die Funktion.
Step Over	Führt einen einzigen Befehl aus.
Step Out	Alle Befehle bis zum Verlassen der Funktion werden abgearbeitet.
Run to Cursor	Führt alle Befehle bis zur Position des Cursors aus.
New Breakpoint	Fügt einen Haltepunkt ein.

2.4.5 Programm testen

Nachdem das Programm erfolgreich übersetzt und übertragen wurde, kann mit dem Testen begonnen werden. Die Ausgabe der LEDs kann einfach durch Zuhilfenahme des eigenen Auges überprüft werden.
Um den Begrüßungstext anzuzeigen wird das *HTerm* Programm verwendet. Es be ndet sich auf der beigelegten CD im Ordner `HTerm`. Zuerst wird der SandboxS 2 „Port“ ermittelt, dies ist die Anschlussnummer der virtuellen seriellen Schnittstelle über die kommuniziert wird. Dazu öffnet man den Geräte-Manager („Start->Ausführen“, „devmgmt.msc“) und sucht nach dem „SandboxS2 serial emulation“ Anschluss (Abbildung 2.20).
Nun kann das Terminalprogramm gestartet und die Verbindung aufgebaut werden. Hierzu wird unter „Port“ die ermittelte Anschlussnummer ausgewählt und anschließend „Connect“ ausgeführt. Zuvor muss die Sandbox natürlich eingeschaltet und vom Betriebssystem erkannt werden.
Nach einem kurzen Druck der Taste Key_Avr0 sollte der Begrüßungstext auf dem Bildschirm erscheinen (Abbildung 2.21).

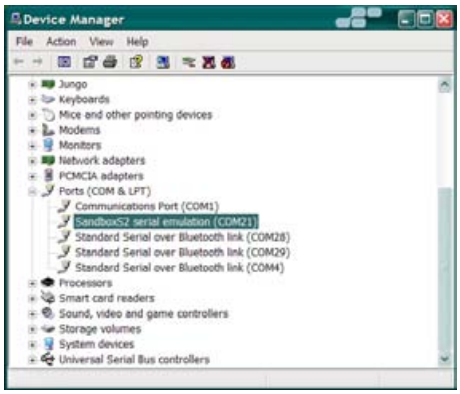


Abbildung 2.20: Geräte-Manager mit „SandboxS2 serial emulation“ Anschlussnummer. Kann mit der Eingabe von „devmgmt.msc“ in den „Ausführen“ Dialog („Start->Ausführen“) aufgerufen werden.

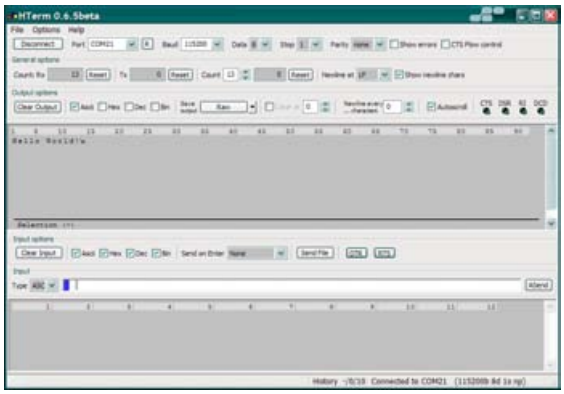


Abbildung 2.21: Terminal Programm HTerm: Alle Einstellungen bis auf die Anschlussnummer sind zu ignorieren. Da des sich um eine virtuelle serielle Schnittstelle handelt, maximale Geschwindigkeit und Flusskontrolle wird durch das USB Protokoll vorgegeben.

2.4.6 Zusammenfassung

In diesem Beispiel wurde die grundlegende Arbeitsweise mit der Hexapod Bibliothek gezeigt. Auch die Verwendung von Datenströme (Streams) wurde beschrieben. Alle Befehle der Hexapod Bibliothek sind in Kapitel 9 aufgelistet. Die restlichen Funktionen sind Bestandteil der *C Standard Bibliothek* und werden im „avr-libc“ Manual, das im WinAVR Paket enthalten ist, beschrieben.

2.4.7 Aufgaben

Diese Aufgaben sollten erfüllt werden, bevor mit dem nächsten Beispiel fortgefahren wird.

a) Ändern Sie den Begrüßungstext auf ihren Namen im Format `Vorname Nachname`.

b) Der Text soll erst nach dem Loslassen der Taste ausgegeben werden.

c) Lassen Sie den Text mit Hilfe einer Schleife nach jedem Tastendruck ausgeben.

2.5 Laufende Lichter

Mit diesem Beispiel soll erstmals der *Robotics Chip* verwendet werden. Die Funktionen des Moduls `RoboticsChip` ermöglichen ein einfaches Arbeiten. Bevor dieses Beispiel begonnen wird, sollte das letzte bereits erfolgreich abgeschlossen sein.

2.5.1 Projekt erstellen

Ein neues Projekt muss erstellt werden. Die Vorgehensweise ist dabei gleich wie in Abschnitt 2.4.1.

2.5.2 Programm schreiben

Im soeben erstellten Projekt kann der Programmcode nun eingegeben werden.

2.5.2.1 Programmspezifikation

Das Programm bringt die LED0 zum Aufleuchten. Einen kurzen Zeitpunkt später erlischt LED0 und LED1 beginnt zu leuchten. Dies wiederholt sich für alle LEDs bis LED11. Danach wird wieder bei LED0 begonnen. Gestartet wird das Lauflicht mit der Taste Key_Avr0. Zusätzlich wird ein Begrüßungstext über die USB Schnittstelle ausgegeben.

2.5.2.2 Quellcode

```
/*****************************************************************************
*  Workfile    : main.c
*  Author      : Christian Raschko
*  Date        : 2007-08-20
*  Description : main program
*  Revision    : 3
*****************************************************************************/
```

```
////////////////////////////////////////////////////////////////////////////
// includes
////////////////////////////////////////////////////////////////////////////

//standard library includes
#include <stdio.h>
//AVR specific includes
#include <avr/interrupt.h>
#include <util/delay.h>
//hexapod library includes
#include "SandboxS2_lib.h"
#include "RoboticsChip.h"
#include "uart.h"
#include "uart_usb_lib.h"


////////////////////////////////////////////////////////////////////////////
// global variables/definitions
////////////////////////////////////////////////////////////////////////////
//pointers to I/O stream structures
FILE * gStrmUSB  = NULL;
FILE * gStrmUART = NULL;


////////////////////////////////////////////////////////////////////////////
// functions
////////////////////////////////////////////////////////////////////////////

//stream I/O wrapper functions
int strm_uart_putc(char c,FILE *s) {uart_putc(c);return 0;}
int strm_uart_getc(FILE *s)        {return (unsigned char)uart_getc();}
int strm_usb_putc(char c,FILE *s)  {uart_usb_putchar(c);return 0;}
int strm_usb_getc(FILE *s)         {return (unsigned char)uart_usb_getchar();}


//do all the initialization stuff here
void Init(void)
{
  //initialize sandboxs2 AVR peripherals
  SL2_Init();

  //initialize robotics chip
  RC_Init();

  //initialize usb
  uart_usb_init();

  //initialize UART, set baud rate to 38400
  uart_init(UART_CALC_BAUDRATE(38400));

  //activate stream I/O
  gStrmUSB = fdevopen(strm_usb_putc,strm_usb_getc);
  gStrmUART = fdevopen(strm_uart_putc,strm_uart_getc);

  //enable interrupts
```

```
  sei();
}


//program entry point
int main (void)
{
  //led number
  uint8_t led = 0;

  //initialize system
  Init();

  //set AVR status led and wait on key
  SL2_AvrSetMsgLed(SL2_MSG_LED_0, SL2_MSG_LED_STATUS_ON);
  SL2_AvrWaitOnKey(SL2_KEY_0, SL2_KEY_PRESSED);

  //print welcome message
  fprintf(gStrmUSB,"Running␣lights!\n");

  //run, light, run
  for(;;)
  {
    //clear all robotics chip LEDs
    RC_SetLedClearAll();
    //set brightness
    RC_SetLedBrightness(led,0xff);
    //delay 160msec
    //maximum delay time is 32.7625msec
    //see delay.h
    _delay_ms(32);
    _delay_ms(32);
    _delay_ms(32);
    _delay_ms(32);
    _delay_ms(32);

    if(led > 11)
      led = 0;
    else
      led++;
  }

  //never return, loop forever
  for(;;);
  return 0;
}
```

2.5.2.3 Bemerkungen

Die Funktionen mit Prä x `RC_` sind Bestandteil des `RoboticsChip` Moduls. Der Robotics Chip ist ein eigenes Stück Hardware, auf der Platine das größte Bauteil.

Die Funktion `_delay_ms()` kehrt erst nach der angegeben Zeit wieder zurück. Dabei gibt es aber Einschränkungen des Wertebereichs, die nicht umgangen werden können. So ist die maximale

Wartezeit auf 32,7625 Millisekunden begrenzt.

2.5.3 Projekt übersetzen

Das fertige Projekt wird nun erstmals übersetzt. Die Vorgehensweise ist dabei gleich wie in Abschnitt 2.4.3.

2.5.4 Programm ausführen

Danach kann das Programm ausgeführt werden. Die Vorgehensweise ist dabei gleich wie in Abschnitt 2.4.4.

2.5.5 Programm testen

Nun wird die Funktion getestet. Nach dem Starten des Programms, und drücken der Taste Key_Avr0, sollte ein Licht von LED0 bis LED11 wandern. Ist es an LED11 angekommen, beginnt es wieder bei LED0.
Die Verwendung des Terminal Programms wurde bereits in Abschnitt 2.4.4 beschrieben.

2.5.6 Zusammenfassung

Da nun die grundlegende Verwendung des Robotics Chips gezeigt wurde, sollten die anderen Funktionen des Moduls ebenfalls betrachtet werden. Hierzu emp ehlt es sich den Abschnitt 9.8 genauer anzusehen. Das Kapitel 7 ist ebenfalls sehr hilfreich, es beschreibt unter anderem die verschiedenen Servo Modi.

2.5.7 Aufgaben

Diese Aufgaben sollten erfüllt werden, bevor mit dem nächsten Beispiel fortgefahren wird.

a) Lassen Sie das Licht in entgegengesetzte Richtung zurück laufen, sobald es an LED11 angekommen ist.

b) Das Licht soll, nach drücken einer Taste, nur einen Schritt laufen.

2.6 Servokalibrierung mit Winkelzeiger

Dieses Beispiel besitzt bereits eine echte Anwendung. Es dient zum Kalibrieren der einzelnen Servomotoren. Dazu wird jedem Servo eine eindeutige Nummer zugeordnet. Über diese können die Kalibrierungswerte danach identi ziert werden. Die Kalibrierung ist notwendig da sich der Winkel nicht linear mit dem Eingangsimpuls ändert.

2.6.1 Projekt erstellen

Die Vorgehensweise ist dabei gleich wie in Abschnitt 2.4.1.

2.6.2 Programm schreiben

Nun kann das Programm geschrieben werden.

2.6.2.1 Programmspezifikation

Beim Einschalten wird der Wert 2048 in das Register PwmLen des Servos 0 geschrieben. Das Servos wird im PwmDirect-Modus betrieben. Die Drehrichtung von Servo 0 kann mittels des Reverse-Bits im Register ModeIrqMask umgekehrt werden. Der Wert im Register PwmLen soll anschließend im Bereich von 0 bis 4095 per Tastendruck eingestellt werden können. Die beiden Tasten Key_Avr0 und Key_Avr1 dienen jeweils dazu, diesen Wert zu erhöhen respektive zu verringern oder die Drehrichtung zu invertieren.

2.6.2.2 Quellcode

```
/*************************************************************************
*  Workfile    : main.c
*  Author      : Christian Raschko
*  Date        : 2007-08-20
*  Description : main program
*  Revision    : 3
*************************************************************************/


////////////////////////////////////////////////////////////////////////
// includes
////////////////////////////////////////////////////////////////////////

//standard library includes
#include <stdio.h>
//AVR specific includes
#include <avr/interrupt.h>
#include <util/delay.h>
//hexapod library includes
#include "SandboxS2_lib.h"
#include "RoboticsChip.h"
#include "uart.h"
#include "uart_usb_lib.h"

//servo port number to use
#define SERVO_TO_USE 0


////////////////////////////////////////////////////////////////////////
// global variables/definitions
////////////////////////////////////////////////////////////////////////
//pointers to I/O stream structures
FILE * gStrmUSB  = NULL;
FILE * gStrmUART = NULL;

//key states
typedef enum
```

```
{
  INCREMENT,
  DECREMENT,
  SELECT,
  NO_KEY
} KeysState;

//servo rotation states
typedef enum
{
  ROT_NORMAL,
  ROT_REVERSE,
} ServoRotationState;


////////////////////////////////////////////////////////////////////////
// functions
////////////////////////////////////////////////////////////////////////

//stream I/O wrapper functions
int strm_uart_putc(char c,FILE *s) {uart_putc(c);return 0;}
int strm_uart_getc(FILE *s)        {return (unsigned char)uart_getc();}
int strm_usb_putc(char c,FILE *s)  {uart_usb_putchar(c);return 0;}
int strm_usb_getc(FILE *s)         {return (unsigned char)uart_usb_getchar();}


//do all the initialization stuff here
void Init(void)
{
  //initialize sandboxs2 AVR peripherals
  SL2_Init();

  //initialize robotics chip
  RC_Init();

  //initialize usb
  uart_usb_init();

  //initialize UART, set baud rate to 38400
  uart_init(UART_CALC_BAUDRATE(38400));

  //clear all robotics chip LEDs
  RC_SetLedClearAll();

  //activate stream I/O
  gStrmUSB = fdevopen(strm_usb_putc,strm_usb_getc);
  gStrmUART = fdevopen(strm_uart_putc,strm_uart_getc);

  //enable interrupts
  sei();
}


//get keys state
KeysState GetKeysState(void)
{
```

```
  //wait until key gets pressed
  while((SL2_AvrGetKey(SL2_KEY_0) == SL2_KEY_NOT_PRESSED)
    && (SL2_AvrGetKey(SL2_KEY_1) == SL2_KEY_NOT_PRESSED));
  //wait some time to debounce key
  _delay_ms(5);

  //evaluate keys:
  //case 1: both keys pressed
  if((SL2_AvrGetKey(SL2_KEY_0) == SL2_KEY_PRESSED)
    && (SL2_AvrGetKey(SL2_KEY_1) == SL2_KEY_PRESSED))
  {
    //wait until keys get released
    while((SL2_AvrGetKey(SL2_KEY_0) == SL2_KEY_PRESSED)
      || (SL2_AvrGetKey(SL2_KEY_1) == SL2_KEY_PRESSED));
    //wait some time to debounce key
    _delay_ms(5);
    return SELECT;
  }
  //case 2: key0 pressed
  else if(SL2_AvrGetKey(SL2_KEY_0) == SL2_KEY_PRESSED)
  {
    //wait until key gets released
    while(SL2_AvrGetKey(SL2_KEY_0) == SL2_KEY_PRESSED);
    //wait some time to debounce key
    _delay_ms(5);
    return INCREMENT;
  }
  //case 3: key1 pressed
  else if(SL2_AvrGetKey(SL2_KEY_1) == SL2_KEY_PRESSED)
  {
    //wait until key gets released
    while(SL2_AvrGetKey(SL2_KEY_1) == SL2_KEY_PRESSED);
    //wait some time to debounce key
    _delay_ms(5);
    return DECREMENT;
  }

  return NO_KEY;
}


//program entry point
int main (void)
{
  //PWM impulse length
  uint16_t pwm = 1 << 11;
  //servo rotation
  ServoRotationState rot = ROT_NORMAL;

  //initialize system
  Init();

  //set AVR status led and wait on key
  SL2_AvrSetMsgLed(SL2_MSG_LED_0, SL2_MSG_LED_STATUS_ON);
  SL2_AvrWaitOnKey(SL2_KEY_0, SL2_KEY_PRESSED);
  SL2_AvrWaitOnKey(SL2_KEY_0, SL2_KEY_NOT_PRESSED);
```

```
//print welcome message
fprintf(gStrmUSB,"Servo␣calibration␣program:\n");
fprintf(gStrmUSB,"Press␣AvrKey0␣for␣PWM␣value␣increment.\n");
fprintf(gStrmUSB,"Press␣AvrKey1␣for␣PWM␣value␣decrement.\n");
fprintf(gStrmUSB,"Press␣AvrKey0␣and␣AvrKey1␣together␣to␣change␣servo␣rotation.\n
    ");

//set servo mode to PWM direct mode
RC_SetServoMode(SERVO_TO_USE,RC_SERVO_MODE_PWM_DIRECT);
//set servo PWM pulse length
RC_SetServoPwmDirect(SERVO_TO_USE,pwm);
//set LED to signal active servo port
RC_SetLedBrightness(SERVO_TO_USE,0xff);

for(;;)
{
  switch(GetKeysState())
  {
    //increment PWM pulse length
    case INCREMENT:
      pwm++;
      fprintf(gStrmUSB,"PWM:␣%u\n",pwm);
      break;

    //decrement PWM pulse length
    case DECREMENT:
      pwm--;
      fprintf(gStrmUSB,"PWM:␣%u\n",pwm);
      break;

    //switch servo rotation
    case SELECT:
      if(rot == ROT_REVERSE)
      {
        fprintf(gStrmUSB,"Rotation:␣normal\n");
        RC_SetServoRotation(SERVO_TO_USE,RC_SERVO_REVERSE_OFF);
        rot = ROT_NORMAL;
      }
      else
      {
        fprintf(gStrmUSB,"Rotation:␣reverse\n");
        RC_SetServoRotation(SERVO_TO_USE,RC_SERVO_REVERSE_ON);
        rot = ROT_REVERSE;
      }
      break;

    default:
      break;
  }
  //apply new PWM value
  RC_SetServoPwmDirect(SERVO_TO_USE,pwm);
}

//never return, loop forever
for(;;);
```

```
  return 0;
}
```

2.6.2.3 Bemerkungen

Die `enum` Typen dienen zur Gliederung der einzelnen Zustände. Dadurch wird die Fehlersuche erleichtert, und der Code ist lesbarer.

Auf der SandboxS 2 be nden sich mehrere Taster, diese sind *reale* Taster. Das bedeutet, sie Prellen beim Drücken und Loslassen. Daher wird mit der Wartefunktion `_delay_ms()` die Zeitspanne, in der Störsignale auftreten können, abgewartet.

Der `++` und `--` Post x der Variable `pwm` stellt eine Abkürzung des Ausdrucks `pwm = pwm + 1` bzw. `pwm = pwm - 1` der Programmiersprache dar.

2.6.3 Projekt übersetzen

Die Vorgehensweise ist dabei gleich wie in Abschnitt 2.4.3.

2.6.4 Programm ausführen

Die Vorgehensweise ist dabei gleich wie in Abschnitt 2.4.4.

2.6.5 Programm testen

Nach dem Starten des Programms, und drücken der Taste Key_Avr0, sollte ein Begrüßungstext ausgegeben werden. Es wird nun ein Servo an den SERVO0 Anschluss angehängt. Danach kann die Position des Servos mit den Tasten verändert werden.
Die Verwendung des Terminal Programms wurde bereits in Abschnitt 2.4.4 beschrieben.

2.6.6 Zusammenfassung

In diesem Beispiel wurde der PwmDirect-Modus des Robotics Chips verwendet. Mit diesem lässt sich ein Servo direkt über PWM Impulse steuern. Der Winkel als Eingabe der PWM Impulslänge ist jedoch nicht linear. Dies soll mit der Sevolinearisierung kompensiert werden. Ebenfalls ist die Standarddrehrichtung zu berücksichtigen, sie kann für jede Servo-Serie anders sein. Die ermittelten Daten werden in Kapitel 8 verwendet.

Nach der erfolgreichen Kalibrierung und Archivierung der Messreihen kann der Hexapod zusammengebaut werden . Nähere Informationen be nden sich im beigelegten *Captain Ahab - Bau-/Mechanikhandbuch*.

2.6.7 Aufgaben

Diese Aufgaben sollten erfüllt werden, bevor das nächste Kapitel begonnen wird.

a) Um den Kalibrierungsvorgang zu erleichtern soll die Geschwindigkeit der Servodrehbewegung, durch längeres Drücken einer Taste, erhöht werden. Erweitern Sie nun ihr Programm um diese Funktionalität.

b) Durch die Beschleunigung der Drehbewegung kann der Wertebereich der Variable `pwm` (0 - 4095) schnell verlassen werden. Implementieren Sie nun zwei Abfragen, die die Werte der Variable `pwm` in den Grenzen 0 bis 4095 halten. Nimmt die Variable den Wert 0 an darf sie nicht mehr decrementiert werden. Nimmt sie den Wert 4095 darf sie nicht mehr inkrementiert werden.

c) Kalibrieren Sie die Servos mit dem nun erstellten Programm gemäß Abschnitt 7.6.1. Nummerieren Sie die Servomotoren und erstellen Sie eine Tabelle die jedem Servo 17 Stützstellen zuordnet. Zusätzlich soll auch die Standarddrehrichtung vermerkt werden, *Reverse* oder *Normal*. Eine Vorlage für Aufkleber zur Servonummerierung ist im *Captain Ahab - Bau-/Mechanikhandbuch*) zu nden.

3 Energieversorgung

Die Energieversorgung der SandboxS 2 nimmt einen relativ großen Bereich der Platine in Anspruch

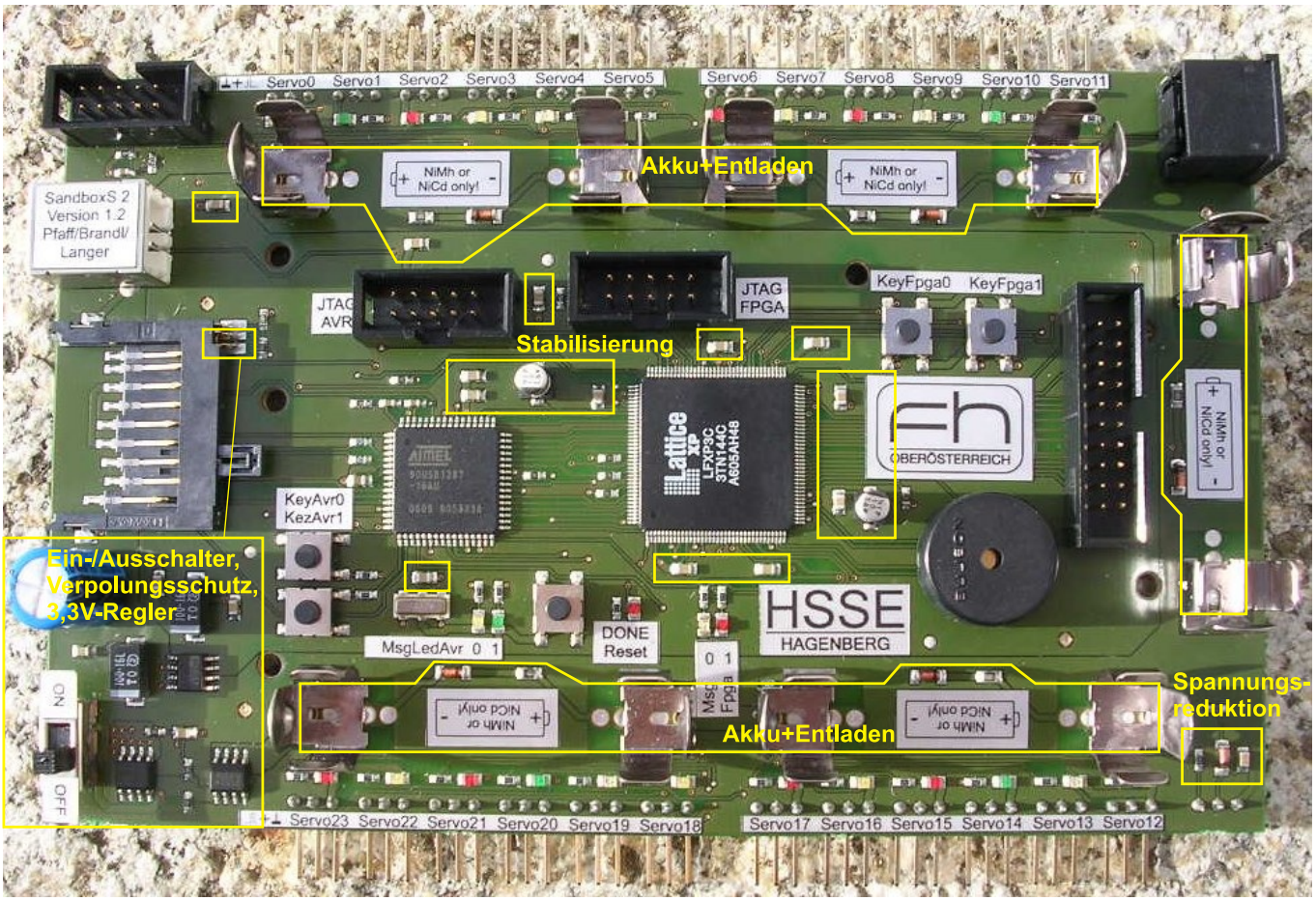


Abbildung 3.1: Für die Energieversorgung verantwortliche Bereiche sind eingerahmt.

(Abb. 3.1). Energiequelle sind fünf handelsübliche Akkuzellen der Größe AA, welche direkt in stabile Halter auf der Platinenoberseite eingelegt werden. Da die Akkuzellen zum Entladen (beseitigt Memory-Effekt) auch verkehrt eingelegt werden können, ist ein Verpolungsschutz vorgesehen. Auch ein Ein-/Ausschalter ist vorhanden, wobei alternativ durch Einstecken einer SD-Karte eingeschaltet werden kann. Auf Grund der hohen Maximalströme der Servos im Bereich von 10 A, wird ein Leistungstransistor zum Schalten der Versorgungsspannung eingesetzt.

Während die Servos unmittelbar von den Akkuzellen versorgt werden, ist für die Elektronikkomponenten eine 3,3-V-Versorgungsspannung erforderlich. Diese stellt ein Linearregler aus der Akkuspannung zur Verfügung.

3.1 Akkuzellen

Als netzunabhängige Energiequelle der SandboxS 2 dienen fünf NiMh- oder NiCd-Zellen der Größe AA. Besonders empfehlenswert ist die Verwendung von NiMh-Zellen der neuesten Generation, die eine Kapazität von über 2000 mAh und eine ausreichende Hochstromfestigkeit aufweisen.

Trockenbatterien mit 1,5 V Nennspannung (z.B. Alkali-Mangan) **eignen sich nicht** zum Betrieb der SandboxS 2! Dies liegt zum einen in der zu hohen Leerlaufspannung solcher Zellen und zum anderen an deren hohem Innenwiderstand.

3.2 Ladegerät

Zum Aufladen der Akkuzellen werden diese aus den Haltern herausgenommen und in einem handelsüblichen Schnellladegerät geladen. Es sollte sich um ein *Schnell*ladegerät (Ladezeit ca. 1 h) handeln, da NiMh-Akkumulatoren, die mit geringem Strom aufgeladen werden, im Betrieb keine hohen Ströme abgeben können. Beispielhaft zeigt Abbildung 3.2 ein kostengünstiges Schnellladegerät für 1 bis

Abbildung 3.2: Preisgünstiges Schnellladegerät Typ C200. Die Ladezeit für bis zu 4 Zellen liegt knapp über einer Stunde.

4 AA- oder AAA-Zellen.

Die Notwendigkeit der Schnellladung ist auch der Grund, warum keine Ladeschaltung mit Stromversorgung aus einer USB-Verbindung zum PC vorgesehen wurde. Diese Zusatzfunktion hätte die SandboxS 2 nicht unwesentlich verteuert, jedoch in der Praxis nur geringen Nutzen gebracht, da so nur niedrige Ladeströme erzeugt werden können.

3.3 Akkuhalter

Die Akkuzellen werden in stabilen Batteriehaltern des Herstellers Keystone auf der Platine xiert und zugleich kontaktiert. In Abb. 3.3 sind diese Halter zu sehen. Die verwendeten Halter weisen einen sehr

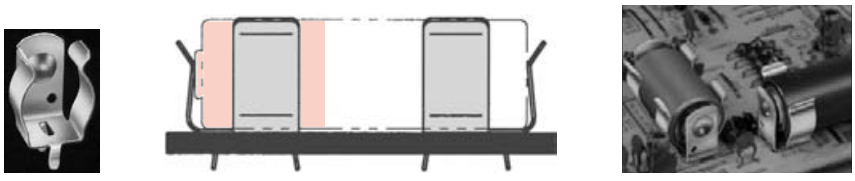


Abbildung 3.3: Halterungen für AA-Zellen von Keystone.

niedrigen Kontaktwiderstand auf und bieten sicheren Halt der Akkuzellen. Der Abstand der Halter-

hälften voneinander kann beim Platinendesign bestimmt werden, wodurch sich der Anpressdruck der Kontaktfedern variieren lässt. Im Falle der SandboxS 2 ist dieser Druck auf ein Maximum eingestellt.

Bitte *verbiegen Sie die Kontaktfedern der Akkuhalter auf keinen Fall*, da deren Federeigenschaften zerstört würden!

Neben dem Anpressdruck der Kontaktfedern spielt deren Oberflächenbeschaffenheit eine wichtige Rolle. Im Falle der Batteriehalter von Keystone kommt eine spezielle Oberflächenveredelung der Rohbleche zum Einsatz. Leider verschlechtert sich der Kontaktwiderstand, sobald Verschmutzungen auftreten. Besonders ungünstig sind Fingerabdrücke. Bitte *berühren Sie also die Kontaktflächen nicht*! Eine Reinigung sollte normalerweise nicht nötig sein. Falls doch, ziehen Sie bei montiertem Akku einfach einen Papierstreifen zwischen den Kontaktflächen der Akkuzelle und des Halters hindurch.

Die **Akkuhalter besitzen keine Isolation und stellen daher ein gewisses Gefahrenpotential dar**. Die Platzierung der Halter verhindert Kurzschlüsse durch Verbiegen. Achten Sie aber stets darauf, dass keine Metallteile auf der Platine liegen (ohnehin keine gute Idee). Ein Stück Draht etwa, das auf die mit Akkuzellen bestückte Platine fällt, verursacht wahrscheinlich einen Kurzschluss der Akkuzellen, die kurzzeitig enorm hohe Ströme (> 40A!) treiben können. Sowohl das Drahtstück, als auch die am Stromfluss beteiligten Teile der Platine, vor allem aber die Akkumulatoren erhitzen sich dabei sehr stark. Brand- und Explosionsgefahr sind die Folge! Die Platine wird in solchen Fällen ebenfalls beschädigt oder gar unbrauchbar. Entfernen Sie vor dem Montieren der Platine auf einem Roboter unbedingt die Akkuzellen aus den Haltern! Gerade von den Schrauben und dem Schraubendreher geht für eine mit Akkuzellen versehene Platine eine große Gefahr aus.

Da auch Halter für AAA-Zellen erhältlich sind, können diese wahlweise bestückt werden. Hierdurch ergibt sich eine Gewichtsersparnis von ca. 60 g, der jedoch ein erheblich ungünstigerer Innenwiderstand der Zellen und deren geringere Kapazität gegenüberstehen. Für Anwendungen mit vielen Servos ergibt sich eine Gesamtstromaufnahme, die mit AAA-Zellen beim derzeitigen Stand der Akkutechnologie nicht befriedigt werden kann.

3.4 Ein-/Ausschalter

Die Versorgungsspannung der SandboxS 2 wird über einen Leistungs-MOSFET der neuesten Generation ein- und ausgeschaltet. Angesteuert wird dieser Transistor sowohl vom Hauptschalter als auch

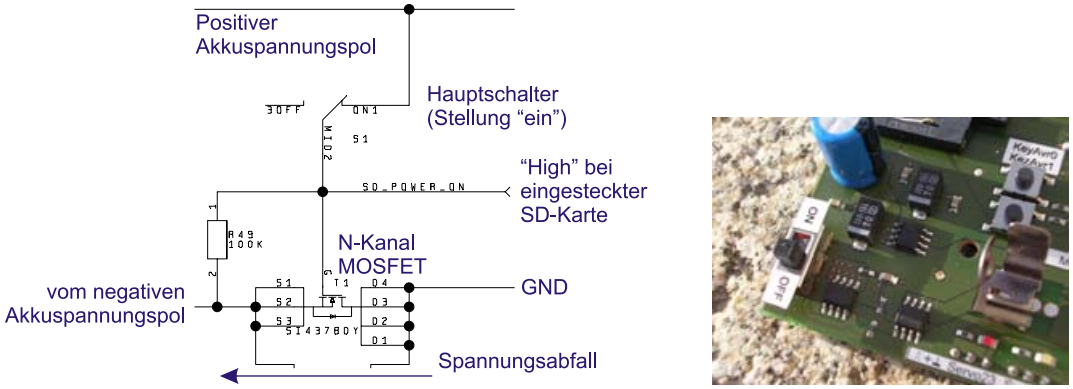


Abbildung 3.4: Ein- und Ausschalten der Spannungsversorgung mittels MOSFET.

durch den SD-CARD-Slot. Sobald der Schalter in die Ein-Position gebracht oder eine SD-Karte ein-

geschoben wird, ist die Versorgungsspannung eingeschaltet. Mit eingesteckter Karte wird daher der Hauptschalter wirkungslos.

3.5 Entladeschaltung

Nur wenige Schnellladegeräte entladen die Akkuzellen vor dem Ladevorgang. Dies ist zur Vermeidung des sogenannten Memory-Effektes durchschnittlich vor jedem fünften Ladezyklus notwendig. Aus diesem Grunde besitzt die SandboxS 2 eine zwar einfache, aber sehr wirkungsvolle Entlade-

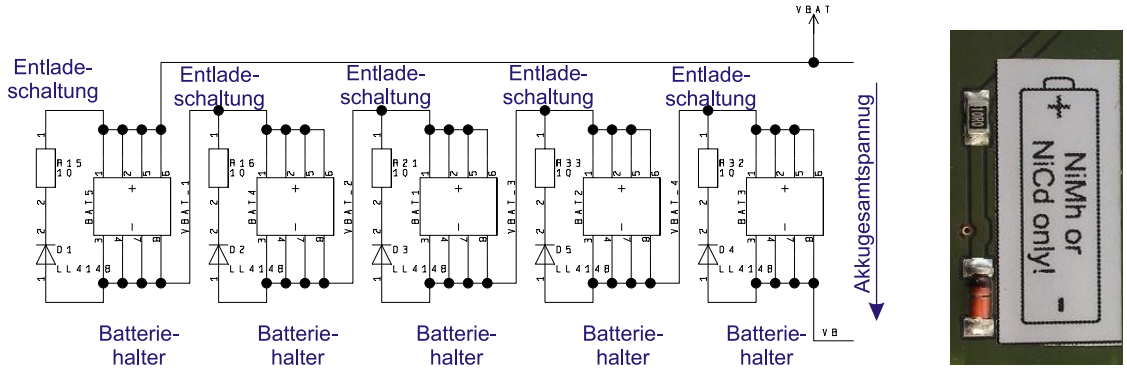


Abbildung 3.5: Entladeschaltung aus Diode und Widerstand.

vorrichtung. Diese besteht aus der Serienschaltung einer Diode und eines Widerstandes (Abb. 3.5). Die Diode lässt einen Stromfluss zu, bis die anliegende Spannung unter die Diodenkniespannung (ca. 0,7 V) sinkt. Der Widerstand begrenzt den fließenden Strom. Sein Wert beträgt 10 Ω. Die in Entladerichtung eingelegte Zelle wird mit einem Strom von

$$I_{\text{Entladen}} = \frac{U_{\text{Akkuzelle}} - U_{\text{Diodendurchlasspannung}}}{R_{\text{Entladen}}} = \frac{1,3 - 0,7\,\text{V}}{10\,\Omega} = 60\,\text{mA}$$

entladen. Bei einer Kapazität von 2500 mAh für einen hochqualitativen NiMh-Akkumulator der Größe AA ergibt sich eine Entladezeit von

$$t_{\text{Entladen}} = \frac{Q_{\text{Akkuzelle}}}{I_{\text{Entladen}}} = \frac{2500\,\text{mAh}}{60\,\text{mA}} = 41,7\,\text{h}$$

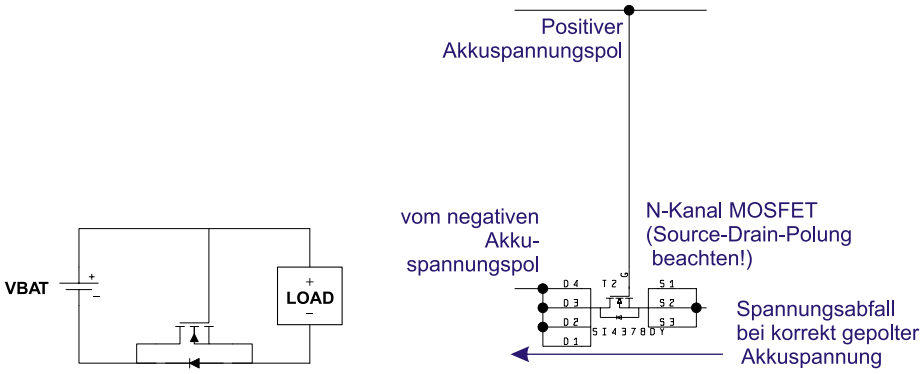
Diese langsame Entladung in ca. 2 Tagen baut den Memory-Effekt gründlich ab.

Die Entladeschaltung wird nur aktiv, wenn eine Akkuzelle verkehrt herum in den Batteriehalter eingelegt wird. Liegt die Akkuzelle in der normalen Richtung im Halter, ist die Diode gesperrt und es fließt kein Strom.

Die Entladeschaltung ist auch für ein auf den ersten Blick merkwürdiges Verhalten verantwortlich, das sich beobachten lässt, wenn vier Akkuzellen bestückt sind und der fünfte Platz leer bleibt. Auch dann arbeiten Mikrocontroller und FPGA bereits. Sobald aber ein Servo bewegt werden soll, versagt das System seinen Dienst und löst selbsttätig einen Reset aus. Diode und Widerstand überbrücken bei fehlender Akkuzelle den leeren Batteriehalter. Die Spannung der verbleibenden vier Akkuzellen (4,8 V-Diodenkniespannung, also 4,1 V) reicht zum Betrieb der 3,3-V-Komponenten aus. Steigt der Strom jedoch an, so fällt am Widerstand (10 Ω) bereits eine nennenswerte Spannung ab. Die Gesamtspannung reicht dann selbst zum Betrieb von FPGA und Mikrocontroller nicht mehr aus.

3.6 Verpolungsschutzschaltung

Da eine negative Spannung zur sofortigen Zerstörung der Elektronikkomponenten der Platine führen würde, ist ein Verpolungsschutz vorhanden. Die in Abb. 3.6 dargestellte Schaltung verwendet statt



(a) Prinzip (b) Realisierung

Abbildung 3.6: Verpolungsschutz für hohe Ströme auf Basis eines Leistungs-MOSFETs. Links die Prinzipschaltung, rechts ein Ausschnitt aus dem Schaltplan der SandboxS 2.

einer Diode einen Leistungs-MOSFET mit niedrigem Drain-Source-Widerstand.

Der Stromfluss durch Drain und Source geschieht im Normalbetrieb in Richtung der internen Freilaufdiode. Das Gate des Transistors ist mit der positiven Akkuspannung verbunden. Bei falscher Polung ist die Spannung V_{GS} negativ, wodurch der Transistor sperrt. Auch die Freilaufdiode leitet nicht. Die Spannung, welche am Durchlasswiderstand des MOSFET abfällt, ist erheblich kleiner als bei einer Diode. Die Verlustleistung sinkt hierdurch ebenfalls.

Eingesetzt wird der MOSFET Si4378DY von Vishay Siliconix. Im Gehäuse SO8 wird ein Durchlasswiderstand von 2,7 mΩ erreicht. Bei einem Strom von $I_{\mathrm{DS}} = 10\,\mathrm{A}$ ergibt sich daher ein Spannungsabfall von 27 mV und ein Leistungsumsatz von 270 mW. Dieser Strom wird aber nur in Extremfällen und auch dann nur kurzzeitig erreicht.

Erst hinter dem Verpolungsschutz dürfen Kondensatoren mit Polung zwecks zusätzlicher Spannungsglättung angeschlossen werden!

3.7 Spannungsregelung

3.7.1 Spannungsversorgung der Elektronikkomponenten

Sowohl der Mikrocontroller als auch das FPGA benötigen eine Spannungsversorgung mit 3,3 V, die durch einen linearen Spannungsregler aus der Akkuspannung bereitgestellt wird (Abb. 3.7). Die Stromaufnahme des FPGAs bewegt sich unter 100 mA, die des Mikrocontrollers unter 15 mA. Hinzu kommen noch die Ströme zum Betrieb der LEDs und der Schnittstellen (SD-Card oder -Modul und PS/2-Maus). Der eingesetzte Spannungsregler kann einen Strom von maximal 300 mA bereitstellen.

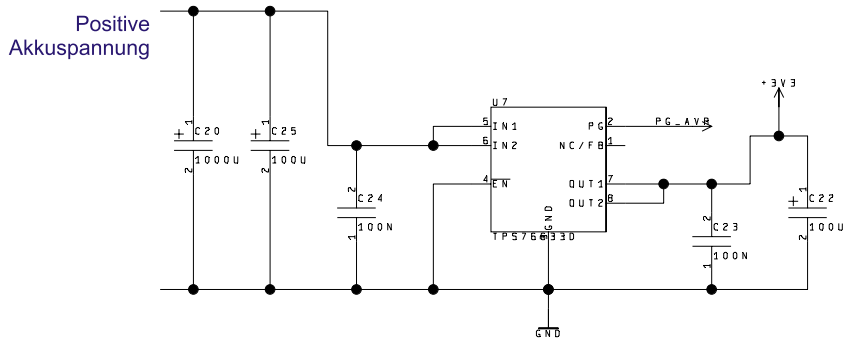


Abbildung 3.7: Spannungsregelung und -stabilisierung.

3.7.2 Spannungsversorgung RC-Empfänger

RC-Empfänger werden traditionell mit 4 AA-Zellen versorgt. Die meisten modernen Empfänger können jedoch auch mit 5 AA-Zellen versorgt werden, da sie einen Spannungsregler enthalten. Um eine sichere Spannungsversorgung von Empfängern jeden Alters zu garantieren, wird, wie in Abbildung 3.8 zu sehen, eine Siliziumdiode in die Versorgung des Empfängers eingeschleift. Sie setzt die Versorgungspannung um 0,7 V herab. Die so erzeugte Spannung wird mittels eines Kondensators stabilisiert. Diese um 0,7 V herabgesetzte Spannung ist zwar ein Kompromiss, hat sich aber in der Praxis gut bewährt.

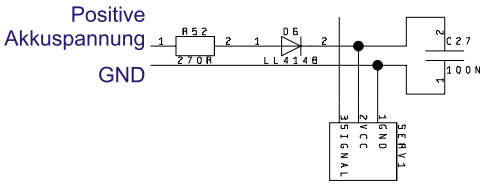


Abbildung 3.8: Versorgungsspannung für den Empfänger.

3.7.3 Spannungsversorgung PS/2-Buchse

Die PS/2-Buchse wird statt aus einer 5 V aus der 3,3 V-Spannungsversorgung des Systems gespeist. Die überwiegende Anzahl der PS/2-Mäuse funktioniert auch mit dieser Betriebsspannung fehlerfrei.

3.7.4 Spannungsversorgung des ADCs im Mikrocontroller

In Abb. 3.9 ist die Spannungsversorgung des im AVR-Mikrocontroller integrierten ADC dargestellt. Die Versorgungspannung AVCC wird aus der 3,3-V-Versorgung über ein RC-Glied, das als Filter

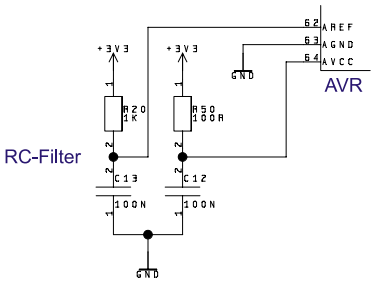


Abbildung 3.9: ADC-Spannungsversorgung und Gewinnung der Referenzspannung mittels RC-Filter.

dient, gewonnen. Die Eigenschaften dieses Filters lassen sich über die Auswahl des Widerstandswertes und der Kondensatorkapazität beeinflussen. Bei zu hohen Widerstandswerten wird jedoch die minimale Versorgungsspannung des ADCs nicht mehr erreicht. In Folge löst der ADC intern im Mikrocontroller unter Umständen einen Reset aus. Der angegebene Wert von 100 Ω hat sich in der Praxis als guter Kompromiss erwiesen.

In der Abbildung ist auch dargestellt, wie die Referenzspannung des ADCs aus der Versorgungsspannung gewonnen wird. Auch hierzu wird ein RC-Filter verwendet. Der eingesetzte Widerstandswert kann um ein Vielfaches höher gewählt werden als bei der Versorgung des ADCs, da in den Referenzspannungseingang des ADCs nur ein minimaler Strom fließt.

4 Mikrocontroller Atmel AVR AT90USB1286/7

Der Mikrocontroller Atmel AVR AT90USB1286/7 wird in einem sehr ausführlichen Datenblatt beschrieben, dass von der WWW-Seite der Firma Atmel geladen werden kann (http://www.atmel.com/dyn/resources/prod_documents/doc7593.pdf).

4.1 Emulation mit dem JTAG ICE Mk II

Zur Emulation kann der Atmel JTAG ICE II verwendet werden (der ältere JTAG ICE kann für den eingesetzten Mikrocontroller nicht benutzt werden). Um diesen Emulator verwenden zu können, wird ein Zugriff auf den JTAG-Port benötigt, der entsprechend Abb. 4.1 auf eine 10-polige Stiftwanne

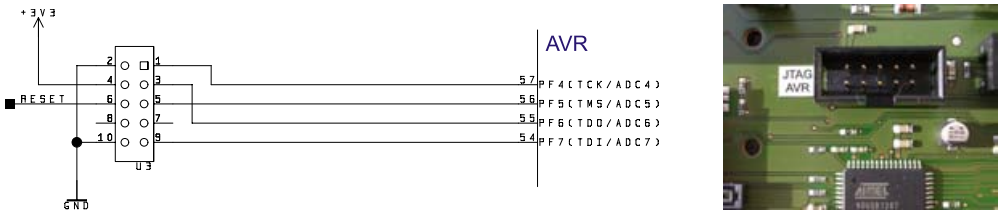


Abbildung 4.1: Der JTAG-Port des AVR ist auf einer 10-poligen Stiftwanne zugänglich.

geführt ist.

Der JTAG-Port ist beim AT90USB1286/7 nicht konform zum JTAG-Standard SOCIETY (1993) ausgeführt, da die verwendeten Anschlusspins eine Zweitfunktion besitzen. Für die Anwendung auf einem mobilen Roboter ist diese Zweitfunktion unter Umständen interessant, da hier die Eingänge ADC4-ADC7 des A/D-Umsetzers verfügbar sind. Die Anschlusswanne für den JTAG ICE II erfüllt also unter Umständen eine Zweitfunktion zum Anschluss von Sensoren.

4.1.0.1 Anschluss des JTAG ICE Mk II an die SandboxS 2

Im Bild4.2 ist der Anschluss des JTAG ICE Mk II an die SandboxS2 dargestellt.

Achtung: Das Folienflachbandkabel des JTAG ICE Mk II ist leider sehr emp ndlich. Bitte achten Sie darauf, dass dieses Kabel nicht auf Zug beansprucht und keinesfalls geknickt wird.

4.2 Taktversorgung

Der Mikrocontroller kann bei 3,3 V Versorgungsspannung maximal mit der Taktfrequenz 8 MHz betrieben werden. Ein Takt dieser Frequenz steht über einen Chip-internen RC-Oszillator zur Verfügung. Jedoch ist dessen Genauigkeit für den Betrieb des integrierten USB-Peripherals nicht ausreichend. Aus diesem Grund wurde ein Quarzresonator in Verbindung mit der AVR-eigenen Oszillatorschaltung zur Erzeugung eines ausreichend frequenzgenauen Taktes vorgesehen (Abb. 4.3)

Abbildung 4.2: Anschluss des Atmel JTAG ICE Mk II an die SandboxS 2. Das andere Ende des USB-Kabels, welches das Bild am oberen Rand verlässt, wird selbstverständlich an den PC angeschlossen.

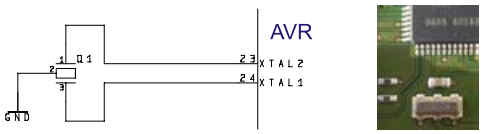


Abbildung 4.3: Quarzresonator zur Frequenzstabilisierung am Mikrocontroller.

Das FPGA wird durch den Takt des Mikrocontrollers ebenfalls versorgt. Dieser legt den (auch internen) Takt auf dem Pin CLKO, sofern die entsprechende *Fuse* des AVR programmiert ist (siehe Abschnitt 4.6). Dies sollte selbstverständlich stets der Fall sein.

4.3 Reset

Am Mikrocontroller gibt es die Möglichkeit, einen Reset mittels eines Tasters zu erzwingen. Der Tas-

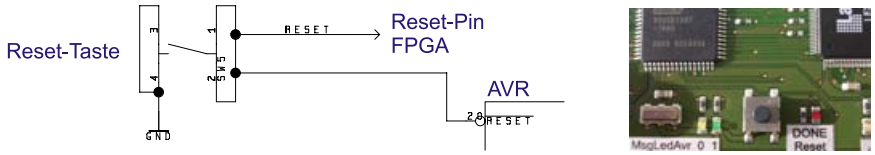


Abbildung 4.4: Reset-Taster für den Mikrocontroller und das FPGA.

ter ist am Anschluss $\overline{\text{RESET}}$ des Mikrocontrollers angeschlossen und legt den Pin im geschlossenen Zustand auf Masse. Im offenen Zustand wird ein interner PullUp-Widerstand des AVR wirksam. Der Taster ist parallel hierzu auch an den Reset-Pin des FPGAs angeschlossen.

4.4 Interrupts

Der HSSE-Robotik-Chip hat die Möglichkeit, über die beiden Interrupt-Leitungen INT0 und INT1 des AVR-Mikrocontrollers einen Interrupt-Request anzufordern. Die genaue Beschreibung dieser Funktion ndet sich im Abschnitt „External Interrupts“ im Datenblatt des Mikrocontrollers.

Der Interrupt kann im Mikrocontroller sowohl pegelsensitiv als auch auf die fallende Flanke der INT-Pins ausgelöst werden. Die Interrupts sind dazu geeignet, den Mikrocontroller aus einem Stromsparmodus (*sleep mode*) aufzuwecken.

Details zur Verwendung der vom HSSE-Robotik-Chip kommenden Interrupts nden sich im Abschnitt 7.2.

4.5 External Memory Interface

Das External Memory Interface des AVRs wird auf der SandboxS 2 genutzt, um die Peripherieeinheiten des HSSE-Robotik-Chips in den Speicherbereich des Mikrocontrollers einzublenden. Hierdurch erscheinen die externen, also im FPGA realisierten Peripherieeinheiten der Mikrocontrollersoftware in der gleichen Art, wie die interne Peripherie des AVRs. Zugleich lässt sich auf diese Weise eine hohe Zugriffsgeschwindigkeit erreichen.

4.6 Einstellung der Fuses des AVR

Die Fuses fabrikfrischer Chips müssen für den Betrieb auf der SandboxS 2 laut Abb. 4.5 gesetzt werden. Die korrekte Einstellung der *Fuses* ist die Basis für die einwandfreie Funktion des AVRs auf der SandboxS 2. Andere als die oben angegebenen Einstellungen sollten nur vorgenommen werden,

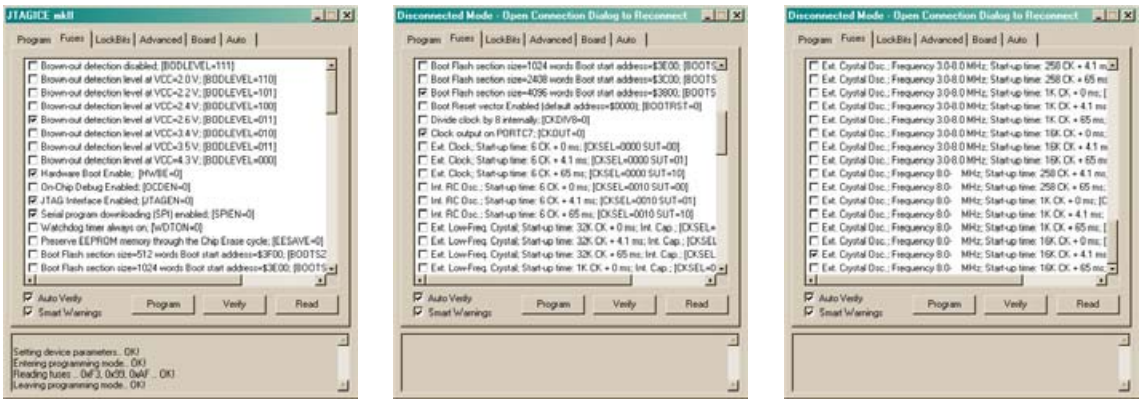


Abbildung 4.5: Screenshots der Fuse-Einstellungen des AVR AT90USB1286/7. Einige Einstellmöglichkeiten für die Taktgenerierung sind nicht dargestellt; diese sind sämtlich nicht ausgewählt (= kein Häkchen).

wenn die entsprechenden Teile des AVR-Datenblatts vollständig gelesen und verstanden wurden. Es ist beispielsweise durch Manipulation mancher der *Fuses* möglich, den weiteren Zugriff vom Debugger JTAG-ICE komplett zu unterbinden. Anschließend lassen sich die *Fuses* nicht mehr verändern und die Platine ist nur noch durch Austausch des AVRs zu retten!

Die Taktversorgung des FPGAs auf der SandboxS 2 erfolgt über den AVR-Mikrocontroller. Damit dieser seinen Takt auf dem entsprechenden Pin ausgibt, muss die Fuse CKOUT programmiert sein (siehe Abb. 4.5).

4.7 Update der Software des AVR-Mikrocontrollers

Die Software auf dem AVR-Mikrocontroller wird in einem Flash-Speicher festgehalten. Dieser kann auf unterschiedliche Weise mit einem neuen Programm versorgt werden. Wichtig sind insbesondere die nachfolgend beschriebenen Vorgehensweisen mittels des USB-Bootloaders und unter Verwendung des *In-Circuit-Emulators* (ICE) JTAG ICE Mk II.

Bei der Programmierung des Speichers mittels USB-Bootloader wird lediglich ein USB-Kabel und die kostenlos unter `http://www.atmel.com` erhältliche Programmierapplikatioon „Flip“ benötigt. Diese Lösung ist daher praktisch kostenlos, bietet aber nicht die Möglichkeiten des Emulators JTAG ICE bei der Unterstützung der Softwareentwicklung. Mit letzterem können vielfältige Informationen über den derzeitigen Zustand von Prozessor und Speicher während des Programmlaufs erfragt und beobachtet werden. Nachteil ist sein relativ hoher Preis.

4.7.1 USB-Bootloader

Der USB-Bootloader für die SandboxS 2 ist ein Programm für den AVR-Mikroccontroller und wird fertig von der Firma Atmel zur Verfügung gestellt. Wird er in den Flash-Speicher des Mikrocontrollers geladen, so belegt er dort die oberen 8 kByte dieses Programmspeichers (insgesamt stehen 128 kByte Programmspeicher zur Verfügung). Man kann daher keine Programme in den Flash-Speicher laden, welche diesen Teil des Speichers ebenfalls belegen, sei es, dass das zu ladende Programm auf Grund seiner Länge in diesen Speicherbereich hineinragt, sei es dass es sich beim zu ladenden Programm um eine Kombination aus einer Applikation und einem eigenen Bootloader handelt.

Beim Debuggen mit dem Debugger JTAG ICE MkII wird der Bootloader leider durch das Ablegen des eigenen Programmes im Flash-Speicher des Mikrocontrollers gelöscht. Ist dies geschehen, so kann der Bootloader wie im Abschnitt 4.7.2 beschrieben wieder in den Flash-Speicher des AVR geladen werden.

Vorbereitung SandboxS 2

Die SandboxS 2 muss folgendermaßen für die Verwendung des USB-Booloaders vorbereitet sein:

- Das Bootloaderprogramm `bl_usb.a90`, auch einfach *Bootloader* genannt, ist in den oberen 8 kByte des Flash-Programmspeichers innerhalb des AVR AT90USB1286/7 abgelegt und
- die Fuse HWBE ist gesetzt (= aktiviert = Häkchen ist gesetzt) (siehe Abschnitt 4.6.

Bei einer frisch gelieferten SandboxS 2 sind beide Voraussetzungen erfüllt. Im Nachhinein lässt sich das Bootloadprogramm `bl_usb.a90` mit in der im Abschnitt 4.7.2 beschriebenen Art an der richtigen Stelle im Flash-Programmspeicher des AVR ablegen. Mit dem dort beschriebenen Werkzeug JTAG ICE Mk II ist auch eine Einstellung der sogennannten *Fuses* und daher auch der *Fuse* HWBE möglich.

Vorbereitung PC

- Auf dem PC muss das Programm „Flip“ installiert sein und
- der Treiber für die den Bootloader auf der SandboxS 2 muss installiert sein.

Das Programm „Flip“, das kostenlos von der WWW-Seite der Firma Atmel (`www.atmel.com`) heruntergeladen werden kann, kommuniziert mit dem *Bootloader* auf der SandboxS 2. Es übermittelt dem *Bootloader*, der vom AVR ausgeführt wird Daten und Befehle, die dieser interpretiert und ausführt. Auf diese Weise kann der Programmspeicher des AVR mit einem Programm gefüllt werden, da der AVR dazu in der Lage ist, seinen eigenen Programmspeicher zu beschreiben. Gleiches gilt auch für den EEPROM-Speicher, der meist für Parameter verwendet wird. Auch dieser kann vom *Bootloader* aus mit Daten versorgt werden.

Die Schritte zur Vorbereitung eines PC sind folgende:

a) Das Programm „Flip“ wird in Form eines Dateiarchivs (`.zip`) von Atmel zur Verfügung gestellt. Dieses Archiv wird zunächst in einem temporären Verzeichnis entpackt.

b) Nun wird die Datei `setup.exe` in diesem Verzeichnis ausgeführt und der *Installer* des Programms „Flip“ beginnt mit seiner Arbeit.

c) Nach abgeschlossener Installation verbindet man die — noch ausgeschaltete — SandboxS 2 an deren USB-Buchse mittels USB-Kabel mit einem USB-Port des PCs.

d) Beim Einschalten der Sandbox wird der *Bootloader* entsprechend der Beschreibung im nachfolgenden Abschnitt 4.7.1 aktiviert.

e) Windows reagiert automatisch mit dem Wizard für die Installation neuer Hardware auf das neue und noch unbekannte USB-Gerät. Die `.inf`-Datei, die Windows für die automatische Treiberinstallation benötigt, wird nicht automatisch vom Wizard gefunden. Diese Datei be ndet sich im Installationsverzeichnis von „Flip“, also beispielsweise unter `C:\Programme\Atmel\FLIP 3.1.0\usb`. Dieses Verzeichnis muss dem Hardware-Wizard manuell bekannt gegeben werden.

f) Im Geräte-Manager (*Device Manager*) ndet sich nun ein Gerät namens Jungo/AT90USB128. Dies sollte man kontrollieren.

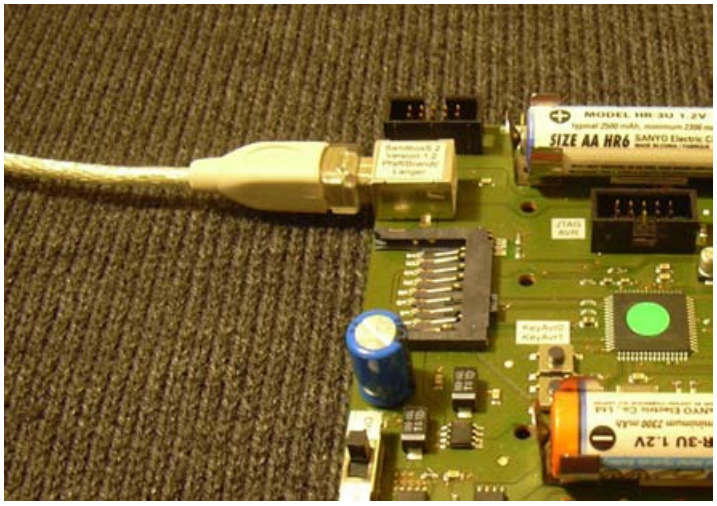


Abbildung 4.6: Anschluss des USB-Kabels an der SandboxS 2. Das andere Ende des Kabels wird an den PC angeschlossen.

Verwendung des Bootloaders

Der Bootloader im AVR wir aktiviert, indem der Reset-Taster auf der SandboxS 2 gemeinsam mit dem Taster KeyFpga0 schon *vor* Einschalten gedrückt und beim Einschalten in dieser Stellung gehalten wird. Nach dem Einschalten wird *zuerst* der Reset-Taster und *danach* der Taster KeyFpga0 losgelassen. Hierdurch wird das Bootloadprogramm aufgerufen. Es arbeitet nach dem immer gleichen Ablauf: Es wartet auf einen Befehl seitens des PCs (z.B. vom Proramm Flip) um ihn unmittelbar nach Ankunft auszuführen. Anschließend wird erneut auf einen Befehl gewartet.

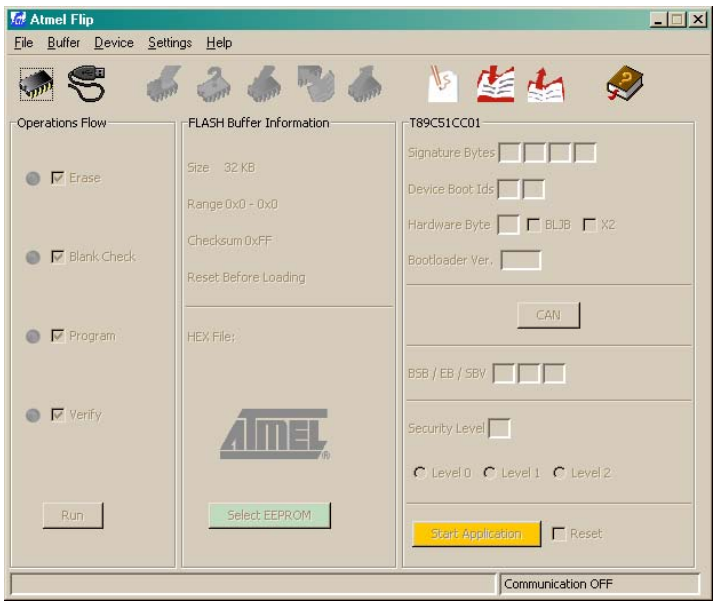


Abbildung 4.7: Das Hauptfenster der Programmierapplikation Atmel „Flip“ (Version: 3.1.1).

Ist der Bootloader im Mikrocontroller aktiv, kann eine Kommunikation mit dem PC aufgebaut werden, wo das Programm „Flip“ mit Hilfe eines Treibers die Gegenstelle bildet. Die abgebildeten

Screenshots des Programms „Flip“ beziehen sich auf die Version 3.1.1. Abbildung 4.7 zeigt das Hauptfenster von „Flip“ unmittelbar nach dem Start des Programmes.

Nun wird „Flip“ für die Verbindung mit der SandboxS 2 kon guriert. Hierzu muss vor allem im Menü *Device – Select...* der richtige Mikrocontroller eingestellt werden. In unserem Fall ist dies der AT90USB1287.

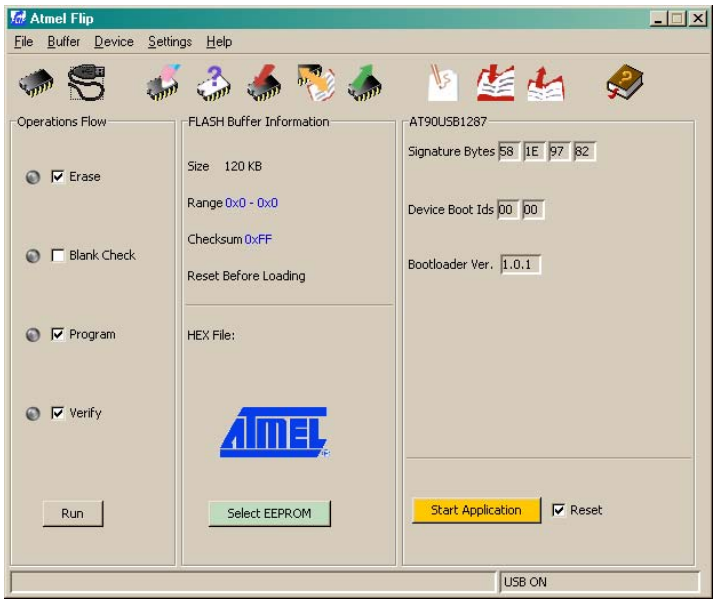


Abbildung 4.8: Der richtige Mikrocontroller ist eingestellt und die Verbindung wurde aufgenommen.

Ist die SandboxS 2 mittels USB-Kabel an den PC angeschlossen und eingeschaltet, so kann mit dem Menüentrag *Settings – Communication – USB* die Verbindung zum Bootloader im AVR-Mikrocontroller aufgenommen werden. Bevor die Verbindung hergestellt wird, erscheint ein Fenster, dessn Button „*Open*“ angeklickt werden muss. Es sollte kein JTAG ICE Mk II gleichzeitig mit dem USB-Kabel an der SandboxS 2 angeschlossen sein. Das Ergebnis der letzten beiden Schritte schlägt sich entsprechend Abbildung 4.8 im Screenshot nieder.

Nun kann eine Programmdatei für den Flash-Speicher des Mikrocontrollers mit dem Menüpunkt *File – Load HEX File...* ausgewählt und geladen werden. Um diese in den Mikrocontroller zu übertragen, kann beispielsweise der Button *Run* links unten im Fenster angeklickt werden. In der Abbildung 4.9 wurde dies für die Datei `TryIrq.hex` getan.

Das Programm be ndet sich nun bereits im Flash-Speicher des Mikrocontrollers. Ausgeführt wird vom AVR aber nach wie vor der Bootloader, der sich ja ebenfalls im Speicher be ndet. Will man das eigene soeben geladene Programm ausführen, gibt es zwei Möglichkeiten: Anklicken des Button *Start Application* oder (besser) Aus- und Einschalten der SandboxS 2.

„Flip“ eignet sich neben der Programmierung des Flash-Speichers im Mikrocontroller auch zur Programmierung des EEPROM-Speichers. Hierzu wird der Button *Select EEPROM* angeklickt. Nun kann im gleichen Verfahren eine entsprechende Datei mit Daten für das EEPROM in den AVR geladen werden.

4.7.2 Software-Update mittels Atmel AVR-Studio und JTAG ICE Mk II

Der In-Circuit-Emulator JTAG ICE Mk II von Atmel kann neben der Emulation auch zur Programmierung von Flash-Speicher und EEPROM des AVR-Mikrocontrollers verwendet werden.

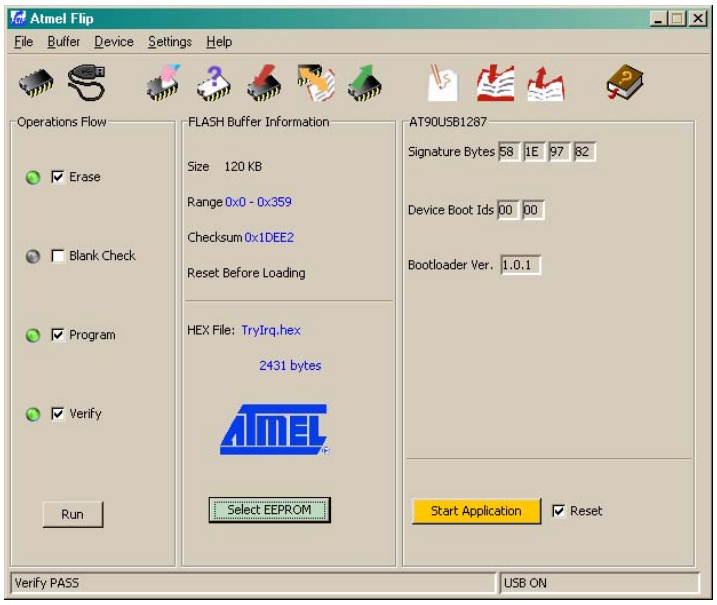


Abbildung 4.9: Der Inhalt der Programmdatei `TryIrq.hex` wurde in den Flash-Speicher des Mikrocontrollers geladen. Die grünen Punkte links neben den Aktionen *Erase, Program, Verify* zeigen an, dass diese erfolgreich durchgeführt wurden.

Hierzu wird im AVR-Studio der Menüpunkt *Tools – Program AVR – Auto Connect* gewählt. Das Fenster des „*AVR Programmer*“ aus Abbildung 4.10 erscheint.

Bootloader und Applikation gemeinsam im Speicher

Im Dialogfenster „*AVR Programmer*“, das vom AVR-Studio aus aufgerufen werden kann, kann die Option „*Erase Device Before Programming*“ deaktiviert werden. Ist dies der Fall, kann eine weitere HEX-Datei in den Flash-Speicher des AVR geladen werden, ohne die bereits dort vorhandene Datei zu löschen. Allerdings sollte gewährleistet sein, dass die beiden Dateien keine überlappenden Speicherbereiche verwenden.

Ist eine der Dateien der Bootloader und die zweite ein Applikationsprogramm, so ist dies normalerweise nicht der Fall, da der Bootloader am oberen Ende des Speichers angesiedelt ist (8 kByte), so dass die unteren 120 kByte für das Applikationsprogramm zur Verfügung stehen.

Vor der Programmierung der ersten Datei, z.B. des Bootloaders sollte allerdings der Flash-Speicher des AVR explizit vom AVR-Programmer aus gelöscht werden (Button „EraseDevice“).

4.8 Atmel AVR Studio und WinAvr

Der eingesetzte Mikrocontroller AT90USB1286/7 benötigt das Atmel AVR-Studio in einer Version nicht älter als 4.13 (`http://www.atmel.com/dyn/products/tools_card.asp?tool_id=2725`) und die Compiler/Debugger-Suite WinAvr (`http://winavr.sourceforge.net/`) nicht älter als Version 20070525.

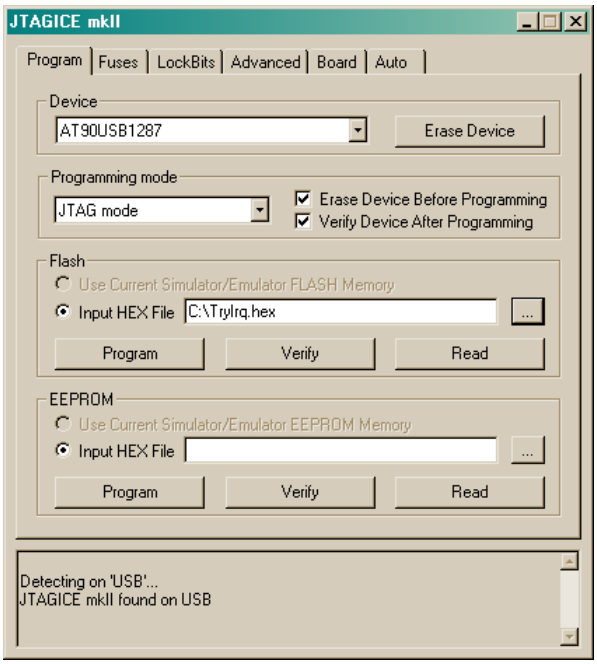


Abbildung 4.10: Der „*AVR Programmer*“, welcher aus dem AVR-Studio heraus gestartet werden kann. Der Baustein AT90USB1287 ist eingestellt und die Datei `TryIrq.hex` ist ausgewählt.

5 On-Chip Peripherie des Atmel AVR AT90USB1286/7

Der Mikrocontroller Atmel AVR AT90USB1286/7 weist eine reichhaltige Auswahl von Peripherieeinheiten auf, die auch zu einem Großteil auf der SandboxS 2 verwendet werden (Abb. 5.1). In die-

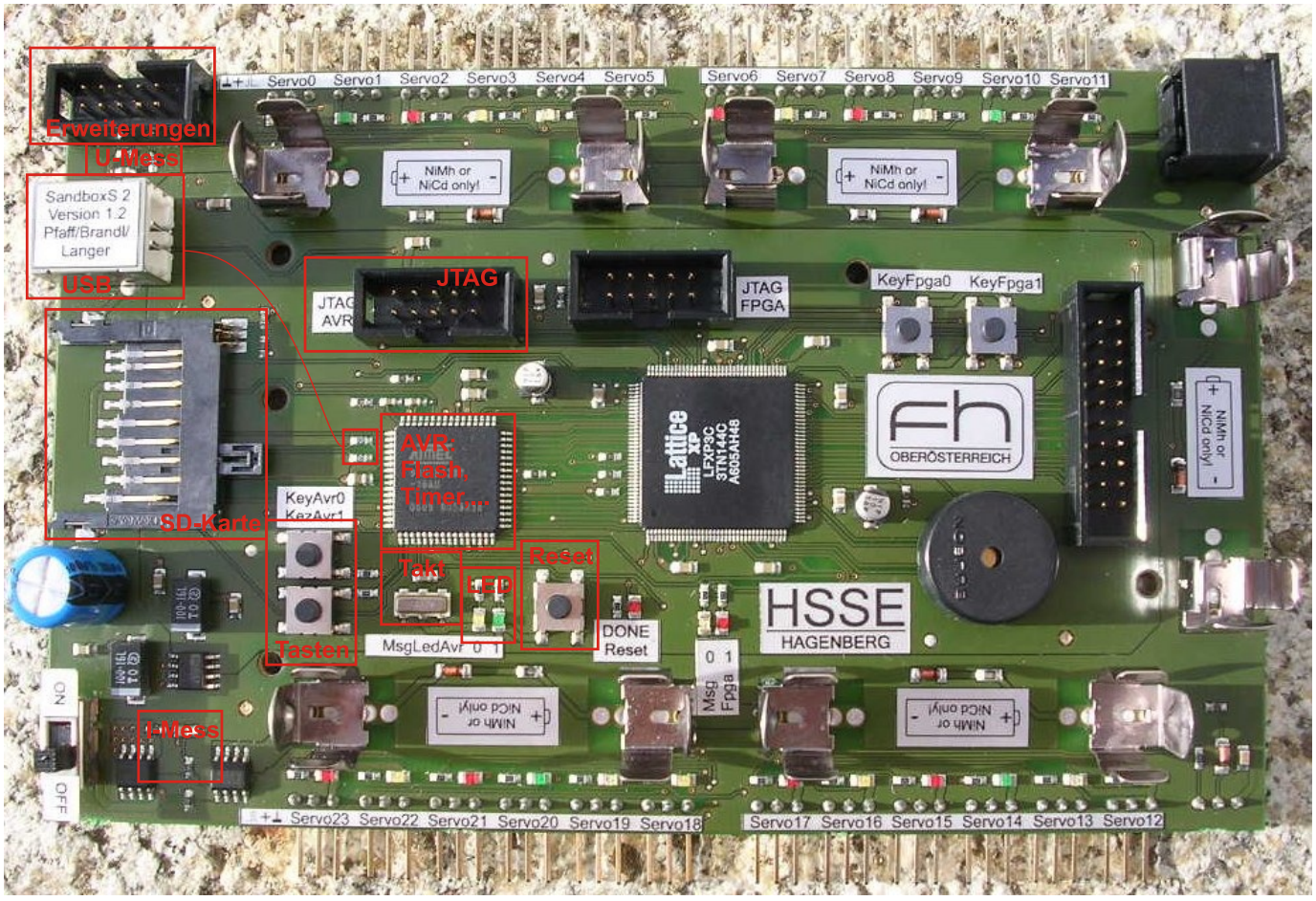


Abbildung 5.1: Der Mikrocontroller Atmel-AVR und seine Peripherieeinheiten sind hervorgehoben.

sem Kapitel werden diese Peripherieeinheiten im Einzelnen vorgestellt. Weitergehende Informationen nden sich im Datenblatt des Mikrocontrollers (`http://www.atmel.com/dyn/resources/prod_documents/doc7593.pdf`) und in den zahlreichen *Application Notes* der Firma Atmel.

5.1 USB-Function Controller

Der Mikrocontroller verfügt über eine eingebaute USB-Schnittstelle nach USB-Standard 2.0 für den *Function*-Betrieb im Modus *Full Speed*. Die USB-Buchse vom Typ B ist über eine minimale Zusatz-

beschaltung laut Abb. 5.2 an den Mikrocontroller angeschlossen. Die Spannungsversorgung, welche

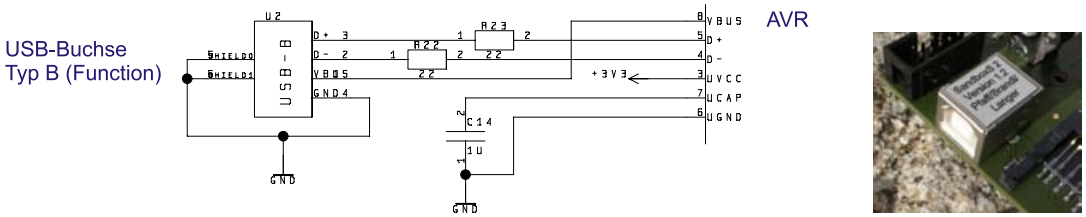


Abbildung 5.2: Anschluss der USB-Buchse vom Typ B an den Mikrocontroller.

vom USB-Host aus über das USB-Kabel möglich wäre wird nur vom USB-Controller des AVR genutzt.

5.2 SD-Card

SD-Cards benötigen eine SPI-Schnittstelle mit nur vier Leitungen und eröffnen neben USB einen zweiten Weg, um Daten mit der Umgebung auszutauschen. Darüberhinaus lassen sich auf SD-Karten große Datenmengen ablegen.

Der Steckplatz für die SD-Card (SD-Card-Slot) ist laut Abb. 5.3 an die SPI-Schnittstelle des AVR

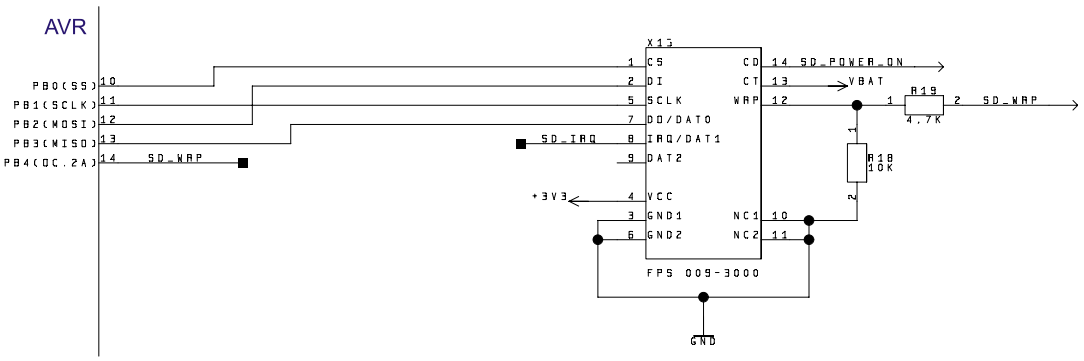


Abbildung 5.3: Anschluss des SD-Card-Slot an die SPI-Schnittstelle des AVR. Das zugehörige Foto ist in Abbildung 5.4 zu sehen.

Abbildung 5.4: Foto des Platinenausschnitts zum Schaltplan in Bild 5.3.

angeschlossen. Der SD-Card-Slot verfügt neben den Anschlüssen zur Kommunikation mit der Karte auch zwei Schaltkontakte, welche

- die Anwesenheit einer Karte und
- die Einstellung des Schreibschutzschiebers der Karte anzeigen.

Der Schaltkontakt zur Anzeige der Anwesenheit einer Karte schaltet die Hauptstromversorgung der SandboxS 2 ein (siehe auch Abschnitt 3.4). Die Stellung des Schreibschutzschiebers der SD-Karte lässt sich über den zweiten Schaltkontakt vom AVR aus einlesen. Selbstverständlich braucht die Bedeutung dieses Schiebers sich nicht auf einen Schreibschutz für die SD-Karte festzulegen. Ebensogut wäre es beispielsweise denkbar, von dessen Stellung das grundsätzliche Verhalten eines Roboters abhängig zu machen.

5.2.0.1 Formatierung der SD-Card unter Windows XP

Die Formatierung der SD-Card sollte *nicht* mit dem in Microsoft Windows integriertem Programm zum Formatieren von Secure Digital Karten erfolgen. Karten, die auf diese Art formatiert wurden sind nicht les- und schreibbar, weil das angelegte Dateisystem mit den Funktionen der Softwarebibliothek nicht voll kompatibel ist.
Um die Speicherkarte dennoch unter Windows zu formatieren, muss wie folgt vorgegangen werden:

□ a

a) Das auf der mitgelieferten CD be ndliche Programm (Pfad?) *SbxRemMedia* starten. Nun bendet sich ein neues Icon (SD-Karte) im Sytem Tray von Windows.

b) Durch einen Rechtsklick auf dieses Icon wird ein Menü geöffnet, in dem „*Format media*“ ausgewählt wird (siehe Abb. 5.5).

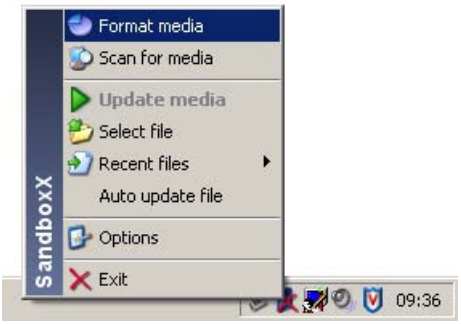


Abbildung 5.5: Systray-Menü des Programms *SbxRemMedia*

c) Zunächst muss das Laufwerk der SD-Karte ausgewählt werden. Wird das entsprechende Laufwerk nicht automatisch angezeigt, so kann dieses manuell über die Option „*Show all drives*“ ausgewählt werden. **Wenn diese Option aktiv ist, muss darauf geachtet werden, dass auch wirklich das richtige Laufwerk gewählt wird.** Abb. 5.6 zeigt die Auswahl des Laufwerks bei einer 128 MB Speicherkarte.

d) Standardmässig erstellt das Programm zwei Partitionen, eine vom Typ *SANDBOXX* und eine vom Typ *HUGE* (siehe Abb. 5.7). Um die SD-Karte mit der SandboxS 2 verwenden zu können, muss die Partition vom Typ *SANDBOXX* gelöscht werden und Partition 0 auf den Typ *FAT_16* umgestellt werden (siehe Abb. 5.8).

e) Nach der zweimaligen Bestätigung mit „OK“ wird das SD-Karten Laufwerk ausgewählt, FAT als Dateisystem eingestellt und die Partition formatiert (siehe Abb 5.9).
Die Karte kann jetzt sowohl von der SandboxS 2 als auch vom PC gelesen (und beschrieben) werden.

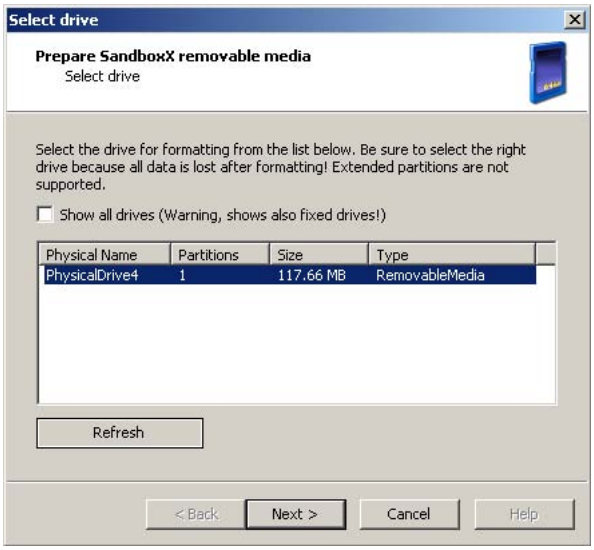


Abbildung 5.6: Auswahl des Laufwerks

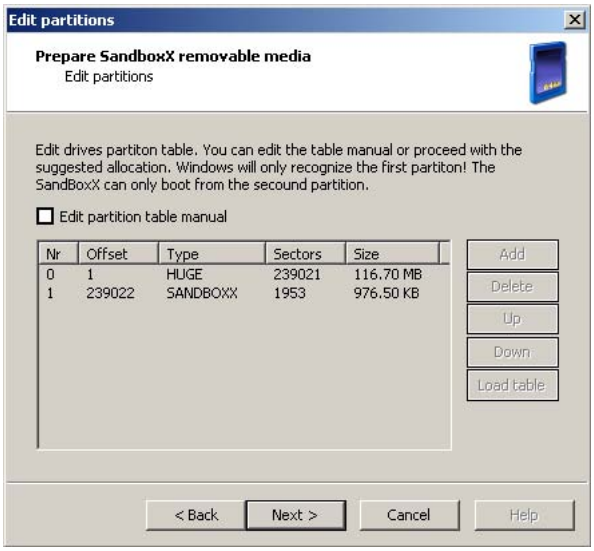


Abbildung 5.7: Standard Partitionierung

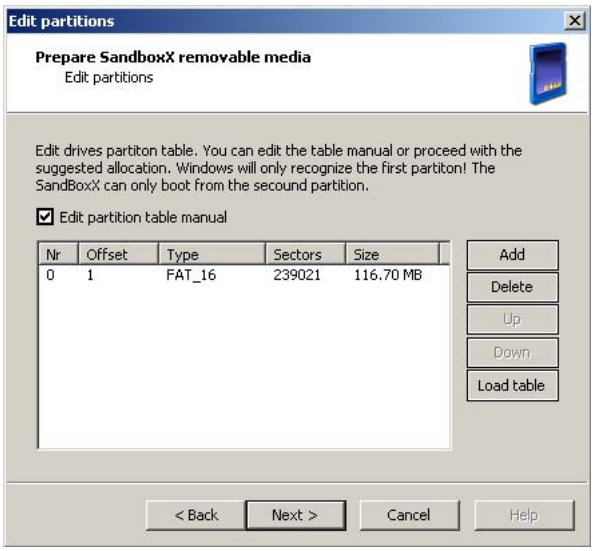


Abbildung 5.8: Auf die SandboxS 2 abgestimmte Partitionierung

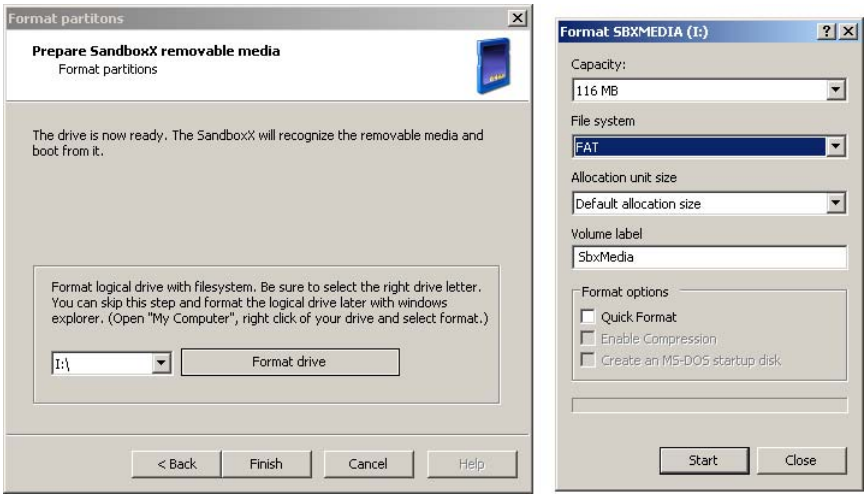


Abbildung 5.9: Formatierung der zuvor erstellten Partition

5.2.0.2 Formatierung der SD-Card unter Linux

Unter Linux gestaltet sich die Formatierung etwas einfacher: Um ein Dateisystem vom Typ FAT16 auf der SD-Karte zu erstellen, wird zunächst mit dem Linux-Befehl `fdisk` eine Partition vom Typ 6 (FAT16) auf der Karte eingerichtet. Danach wird diese Partition mit `mkdosfs` formatiert.

Wer kein Linux installiert hat und dies möglicherweise auch nicht möchte, kann hierzu eine sogenannte Live-CD verwenden. Gut eignet sich beispielsweise Knoppix (`http://www.knoppix.org/`), das kostenlos zum Download verfügbar ist. Die aktuelle Ubuntu-Linux-Distribution ist ebenfalls als Live-CD verwendbar.

5.3 Stromaufnahme und Akkuspannung

Zur Strom- und Spannungsmessung wird der ADC des AVR-Mikrocontrollers verwendet. Dieser besitzt neben der Messung von Spannungen gegenüber dem Bezugspotential GND auch Möglichkeiten um differentielle Messungen mit einstellbarer Verstärkung um den Faktor 1, 10 oder 200 durchzuführen.

5.3.1 Strommessung

Der ADC des Mikrocontrollers erlaubt die Messung der Gesamtstromaufnahme des Systems (SandboxS 2 und Servos), wobei der Restwiderstand des Ein-/Ausschalt-MOSFETs als Shunt-Widerstand genutzt wird. Der Anschluss des Transistorkanals (Drain-Source-Strecke) ist in Abb. 5.10 dargestellt. Der AVR verfügt über differentielle ADC-Eingänge, so dass unter Umgehung des von hohen Strömen

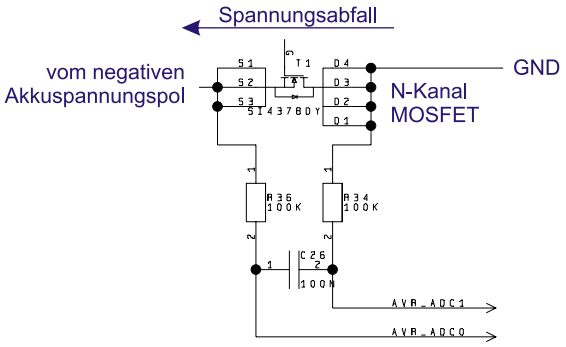


Abbildung 5.10: Die Messung der am Transistorkanal abfallenden Spannung erlaubt den Rückschluss auf den Strom, der über den Kanal fließt.

durchflossenen Weges über GND eine genauere Messung möglich ist.

Dieses Verfahren bietet zwar nicht die Genauigkeit eines speziellen Mess-Shunts, aber benötigt im Gegenzug kaum Platinenfläche und verursacht nur minimale Kosten.

Die am Kanal des Transistors abfallende Spannung bei max. 3 mΩ Kanalwiderstand im eingeschalteten Zustand bewegt sich im Bereich von 0-30 mV. Bei 200-facher Verstärkung ergeben sich daher 0-6 V, während eine zehnfache Verstärkung 300 mV ergibt. In den meisten Fällen wird die 200-fache Verstärkung also verwendet werden können. Bei Überlauf des ADC-Messergebnisses kann einfach eine zweite Messung mit reduzierter Verstärkung erfolgen, da die Verstärkung per Software einstellbar ist.

5.3.2 Spannungsmessung

Der ADC des AVRs wird neben der Strommessung auch zur Messung der Akkuspannung herangezogen. Die entsprechende Schaltung ist in Abb. 5.11 zu sehen. Die Spannung wird über einen Span-

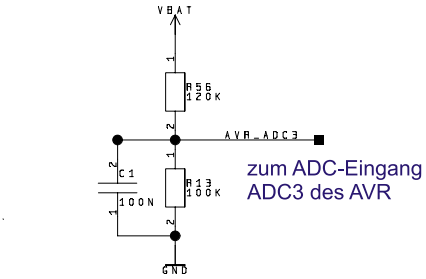


Abbildung 5.11: Schaltung zur Aufbereitung der Spannung für die Messung durch den ADC des AVR.

nungsteiler auf den Messbereich des ADCs im AVR herabgeteilt, da dieser nur von 0-3,3 V geht, die Akkuspannung jedoch unmittelbar nach dem Aufladen knapp über 7 V liegen kann. Das Messprinzip des ADC im Mikrocontroller erfordert einen geringen Ausgangswiderstand der Quelle. Diese wird jedoch nur kurzzeitig belastet. Daher kann ein Kondensator zur Stabilisierung herangezogen werden.

Da die Akkuspannung im laufenden Betrieb gemessen wird, ergibt sich die Möglichkeit, die Entladekurve der Akkus aufzuzeichnen und den Zusammenhang zum dem entnommenen Strom festzuhalten.

5.4 Taster

Am Mikrocontroller sind zwei Tasten laut Abb. 5.12 zur Auslösung beliebiger Aktionen angeschlos-

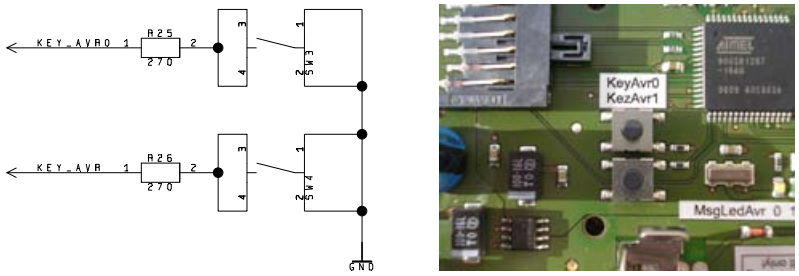


Abbildung 5.12: Universell einsetzbare Taster am AVR.

sen. Die Tasten sind über einen Serienwiderstand als Schutzschaltung angeschlossen, so dass ein Tastendruck auch bei fehlerhafter Programmierung des AVRs keine Zerstörung des Bausteins zur Folge hat.

Um die Stellung einer Taste auswerten zu können, ist der interne Pull-Up-Widerstand des entsprechenden AVR-Ports zu aktivieren. Solange die Taste nicht gedrückt und der Kontakt demnach geöffnet ist, wird vom AVR ein 1-Bit eingelesen. Drückt man die Taste, so liegt das Potential GND am Port, so dass ein 0-Bit eingelesen wird.

5.5 Status-LED

Am Mikrocontroller stehen zwei LEDs (siehe Abb. 5.13 zur Anzeige beliebiger Statusmeldungen zur

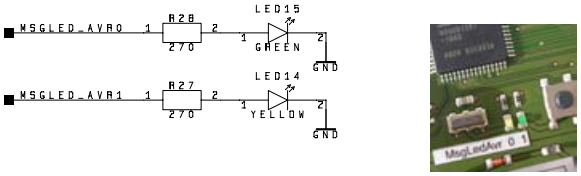


Abbildung 5.13: LED am Mikrocontroller zur Anzeige von Statusmeldungen.

Verfügung. Die Anode jeder LED ist über einen Widerstand zur Strombegrenzung an einen Port des AVRs angeschlossen. Der Port wird als Ausgang programmiert. Gibt man auf dem Port ein 1-Bit aus, so leuchtet die LED, während Sie bei Ausgabe einer 0 dunkel bleibt.

Stellt man den Port als Eingang ein und aktiviert den zugehörigen Pull-Up-Widerstand, so kann man hierdurch unter Umständen ein schwaches Glimmen der LED erzielen. Dessen Intensität hängt stark vom Typ der bestückten LED und sogar von Serienstreuungen in der LED-Fertigung ab. Es kann daher durchaus sein, dass die LED auf diese Weise überhaupt nicht zum Leuchten gebracht werden kann.

5.6 Erweiterungsstecker

Auf dem AVR-Erweiterungsstecker in Form einer 10-poligen Stiftwanne im 2,54-mm-Raster (dargestellt in Abb. 5.14) ist sowohl die Batteriespannung als auch die geregelte 3,3-V-Versorgungsspannung

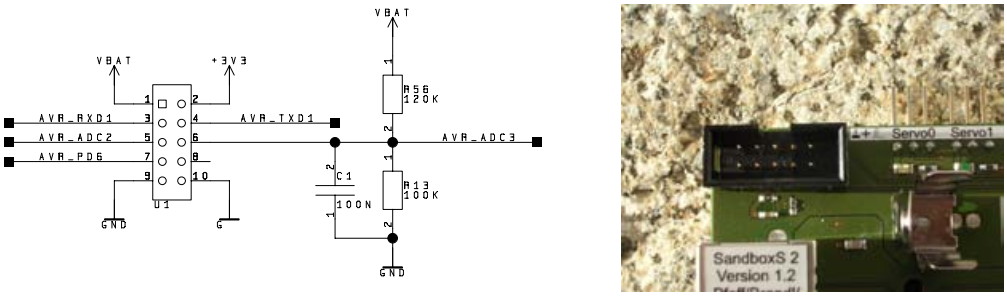


Abbildung 5.14: Erweiterungsstecker (10-poliger Wannensteckverbinder) am Mikrocontroller.

verfügbar. Der GND-Anschluss ist auf zwei Pins vorhanden. Auf dem Steckverbinder stehen folgende Anschlüsse des Mikrocontrollers zur Verfügung:

PD2/RXD Port-Anschluss PD2 oder Eingang der asynchronen seriellen Schnittstelle des AVRs

PD3/TXD Port-Anschluss PD3 oder Ausgang der asynchronen seriellen Schnittstelle des AVRs

PF2/ADC2 Port-Anschluss PF2 oder Eingang ADC2 des ADCs im AVR

PD6/T1 Port-Anschluss PD6 oder *Timer/Counter Clock Input* T1

PF3/ADC3 Port-Anschluss PF3 oder Eingang ADC2 des ADCs im AVR. Bitte Vorbelegung mit der herabgeteilten Akkuspannung beachten. Falls notwendig, können C1, R13 und R56 von der Platine entfernt werden.

Der Pin des Erweiterungssteckers mit der Nr. 8 ist nicht angeschlossen und kann daher frei belegt werden. Hierzu wird er mittels einer Drahtbrücke (*Wire-Wrap*-Draht) mit dem gewünschten Pin des AVRs verbunden.

5.7 Anschluss eines Bluetooth-RS323-Moduls

An die Erweiterungsschnittstelle ist ein Bluetooth-RS232-Modul – auch als Bluetooth-Serial-Modul bezeichnet – anschließbar. Ein solches Modul ermöglicht die drahtlose Steuerung mittels eines PC, insbesondere mittels eines Laptops mit eingebautem Bluetooth-Transceiver.

Das Bluetooth-Serial-Modul wird an die serielle Schnittstelle des AVR-Mikrocontrollers angeschlossen, die auf dem Erweiterungssteckverbinder nach Abschnitt 5.6 zur Verfügung steht. Auch die Spannungsversorgung des Moduls mit 3,3 kann aus diesem Steckverbinder bezogen werden.

Nach Kon guration des Moduls und Verbindungsaufnahme mit dem Bluetooth-System des PCs steht eine drahtlose serielle Schnittstelle (z.B. COM40) zur Verfügung, die auf der PC-Seite als Ein- und Ausgabeport für Daten verwendet werden kann. Diese Daten werden in der gleichen Art zur SandboxS 2 übermittelt, wie dies durch ein serielles Kabel geschieht.

Bluetooth-Serial-Modul Parani ESD 100 von SENA

Zum Anschluss kommen die Module unterschiedlicher Hersteller in Frage. Als Beispiel wird hier das Modul ESD 100 des Hersteller SENA (`http://www.sena.com/products/industrial_bluetooth/oem_bluetooth_serial/parani_esd/`) herangezogen, das zu einem Preis von ca. 50 € innerhalb Europas erhältlich ist. Ein Foto des Moduls ist in Abbildung 5.15 zu sehen.

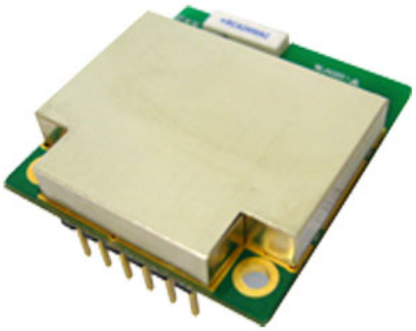


Abbildung 5.15: Das Bluetooth-Serial-Modul Parani ESD 100 der Firma SENA.

Damit das Modul verwendet werden kann, muss es zunächst entsprechend kon guriert werden. Hierzu werden vom Hersteller ein Starterkit (ESD Startkerkit) zum Anschluss des Moduls an den PC und ein Kon gurationsprogramm namens ParaniWIN geliefert. In Abbildung 5.16 sind Kon gurationseinstellungen zu sehen, die sich in der Praxis bewährt haben. Da der AVR mit einem Takt von 8MHz läuft und der Baudrate-Generator diesen Takt verwendet, ergeben sich Synchronisationsfehler. Diese lassen eine Übertragung nur mehr mit starken Fehlern zu. Dadurch wird die Auswahl auf folgende Baudraten beschränkt: 2400, 4800, 9600, 19200, 38400.

Zum Anschluss des Moduls an den Erweiterungsstecker (Abschnitt 5.6) der SandboxS 2 kann beispielsweise ein speziell angefertigtes „Flachband“kabel dienen, dessen Verbindungen nach Tabelle 5.1 angelegt wurden. Bei der Herstellung des Kabels ist die gleiche Technik hilfreich, die auch beim Anschluss des Playstation-GamePad-Kabels verwendet wird (siehe Abschnitt 7.17.1).

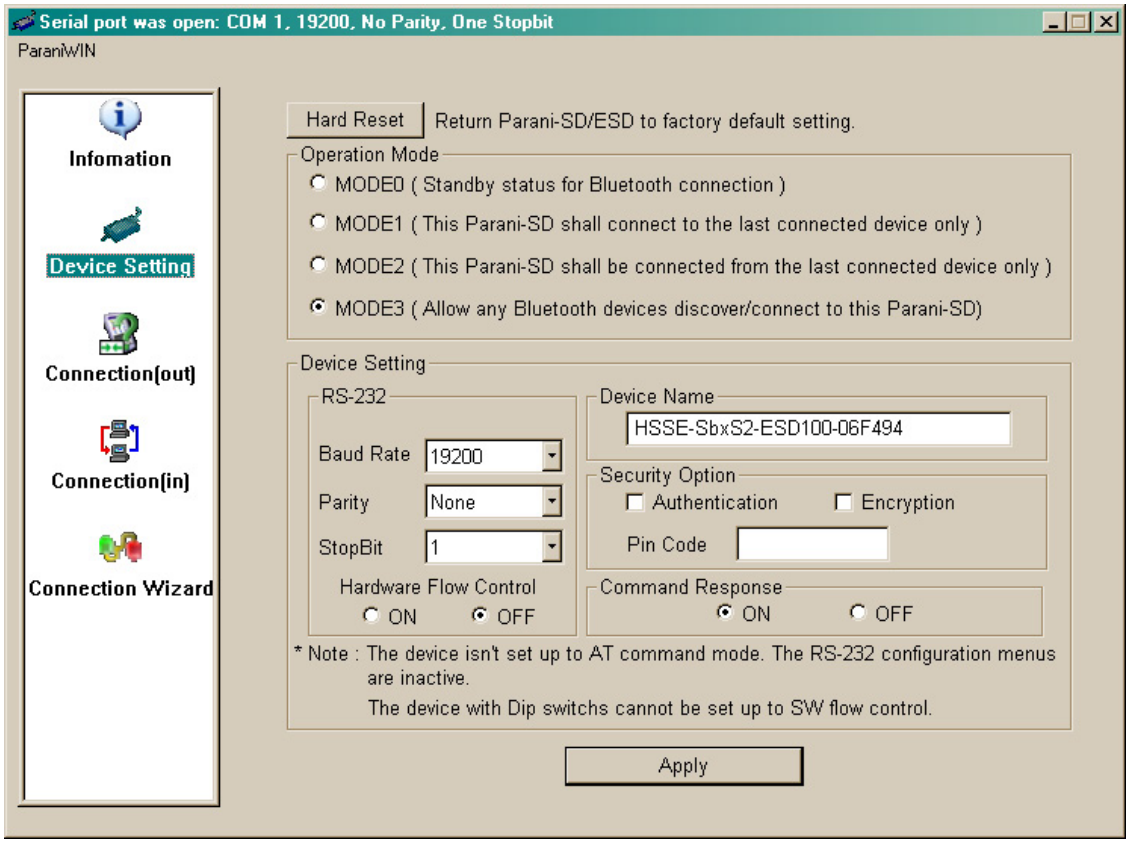

Serial port was open: COM 1, 19200, No Parity, One Stopbit
ParaniWIN
Infomation
Device Setting
Connection(out)
Connection(in)
Connection Wizard
Hard Reset
Return Parani-SD/ESD to factory default setting.
Operation Mode
MODE0 (Standby status for Bluetooth connection)
MODE1 (This Parani-SD shall connect to the last connected device only)
MODE2 (This Parani-SD shall be connected from the last connected device only)
MODE3 (Allow any Bluetooth devices discover/connect to this Parani-SD)
Device Setting
RS-232
Baud Rate 19200
Parity None
StopBit 1
Hardware Flow Control
ON OFF
Device Name
HSSE-SbxS2-ESD100-06F494
Security Option
Authentication Encryption
Pin Code
Command Response
ON OFF
* Note : The device isn't set up to AT command mode. The RS-232 configuration menus are inactive.
The device with Dip switchs cannot be set up to SW flow control.
Apply

Abbildung 5.16: Kon guration des Bluetooth-Serial Moduls Parani-ESD100 des Herstellers SENA unter Verwendung von dessen Kon gurationsprogramm ParaniWIN. Das Modul ist über das ESD StarterKit von SENA an eine serielle Schnittstelle des PC angeschlossen.

Pin	At90USB1286/7	Parani ESD 100	Pin
2	+3V3	VDD	6
3	AVR_RXD1	TXD	2
4	AVR_TXD1	RXD	3
5*	AVR_ADC2	RTS	4
7*	AVR_EXT_PIN	CTS	5
9*	GND	DSR	9
10	GND	GND	12

Tabelle 5.1: Verbinden des Erweiterungssteckers des AVR-Mikrocontrollers auf der SandboxS 2 mit dem Bluetooth-Serial-Modul Parani ESD 100. Die Steckernummerierung des Bluetooth-Moduls folgt nicht der Konvention bei der Nummerierung von Stiftwannen und muss daher dem Datenblatt entnommen werden! *Verbindung 5,7,9 werden nur für Hardware Flusskontrolle benötigt, und können weggelassen werden, falls sie nicht verwendet wird.

6 FPGA Lattice XP LFXP3K

Das verwendete FPGA gehört zur FPGA-Familie Lattice XP, die in einem sogenannten *Handbook* genau beschrieben wird (http://www.latticesemi.com/products/fpga/xp). Die Lektüre dieses Handbuchs ist jedoch nur dann wichtig, wenn ein eigener Chipentwurf auf dem FPGA implementiert werden soll. Wird das FPGA in seiner Rolle als HSSE-Robotics-Chip verwendet, dann genügt die vorliegende Beschreibung.

6.1 Taktversorgung

Das FPGA wird vom Taktgenerator des Mikrocontrollers mitversorgt. Daher arbeiten FPGA und Mikrocontroller taktsynchron bei einer Taktfrequenz von 8 MHz[1].

Der Mikrocontroller gibt den Takt für das FPGA auf dem Pin CLKO jedoch nur dann aus, wenn die entsprechende *Fuse* des AVRs programmiert ist. Dies ist insbesondere bei fabrikneuen Mikrocontrollerchips nicht der Fall. Auf der SandboxS 2 wird diese *Fuse* jedoch unmittelbar nach der Fertigung korrekt programmiert. Näheres zur Einstellung der *Fuses* ndet sich im Abschnitt 4.6.

6.2 Reset

Die Resetleitung des FPGAs ist an den gleichen Taster angeschlossen wie die Resetleitung des Mikrocontrollers. Anders als bei den Peripherieeinheiten des AVRs werden die Register der Peripherieeinheiten, welche das FPGA zur Verfügung stellt, bei einem Reset nicht verändert. Nur ein Aus-Einschaltvorgang setzt diese Werte wieder auf den Initialisierungszustand zurück.

Ausgenommen von dieser Regel sind lediglich die Interrupt-Flag-Register (Abschnitt 7.2). Diese werden durch einen Reset gelöscht, also mit Bits des Wertes 0 gefüllt.

6.2.1 Hardware-Bootloader-Aufruf des AVRs

Wird vor Loslassen der Reset-Taste die Taste KeyFpga0 gedrückt und erst nach dem Loslassen der Reset-Taste wieder losgelassen, so ruft der HSSE-Robotics-Chip die Hardware-Boot-Funktion (HWB) des AVR-Mikrocontrollers auf (Aktivieren der Leitung ALE mit dem Deaktivieren des Reset-Impulses). Soll diese Funktion schon beim Einschalten aufgerufen werden, so sind bereits vor dem Einschalten beide Tasten zu drücken. Man lässt dann zuerst die Reset-Taste und dann die Taste KeyFpga0 los.

Der Mikrocontroller ruft als Reaktion statt des normalen Applikationsprogramms ein Bootloaderprogramm auf. Voraussetzung ist, dass die Fuse HWBE des AVRs entsprechend programmiert ist. Dies ist normalerweise jedoch nicht der Fall (siehe Abschnitt 4.6). Weitere Informationen gibt das Datenblatt des AT90USB1286/7, wenn man es nach dem Stichwort „HWB“ durchsucht.

[1] Falls das FPGA mit einer eigenen Kon guration betrieben werden soll, ist zu beachten, dass die Mindestfrequenz zur Versorgung der FPGA-internen PLLs 25 MHz beträgt. Daher kommt deren Einsatz bei Verwendung eines Taktes von 8 MHz Taktfrequenz nicht in Frage.

6.3 FPGA-Konfiguration

Der Entwurfsprozess für den HSSE-Robotics-Chip endet in der Erzeugung einer sogenannten Kon- gurationsdatei für das FPGA. Diese Datei enthält quasi Schaltereinstellungen, welche die Struktur und die Elementarfunktionen der Schaltung im FPGA bestimmen. Das eingesetzte FPGA vom Typ Lattice XP speichert diese Kon guration in statischen Speicherzellen ab, welche über den gesamten Chip verteilt sind. Daher wird die Kon guration mit jedem Ausschalten gelöscht und steht nach dem Einschalten nicht mehr zur Verfügung. Das FPGA vergisst sozusagen seine Kon guration. Bausteine der Lattice-XP-Serie verfügen jedoch zusätzlich über einen Flash-Speicher, in dem die Kon gurationsdatei abgelegt werden kann.

Auf der SandboxS 2 ist das FPGA so eingestellt, dass es den Inhalt des Flash-Speichers unmittelbar nach dem Einschalten automatisch in die RAM-Kon gurationszellen lädt. Dies benötigt nur wenige Millisekunden, während der die LED 13 in unmittelbarer Nähe des FPGA-Bausteins eingeschaltet

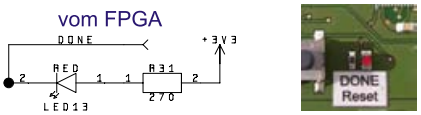


Abbildung 6.1: Die DONE-LED leuchtet, solange das FPGA keine Kon guration geladen hat.

ist. Das kurze Aufblitzen dieser LED beim Einschalten markiert also den Übertragungsvorgang der Kon gurationsdaten.

6.3.1 Update der FPGA-Konfiguration

Der Flash-Speicher des FPGAs kann selbstverständlich mehrfach programmiert werden, wodurch ein Update des HSSE-Robotics-Chips möglich wird. Es ist auch möglich, ein eigenes Hardwaredesign in das FPGA einzuspielen. Um Missverständnissen vorzubeugen, nochmals der Hinweis, dass es sich bei der eingespielten Datei nicht um ein Programm im Sinne von Software (vergleichbar mit einer exe-Datei oder einer hex-Datei) handelt. Diese Datei enthält vielmehr Informationen, welche unmittelbar die Schaltung im Innern des FPGAs bestimmen.

In den folgenden Abschnitten wird besprochen, welche Möglichkeiten es gibt, um die Kon gurationsdatei in das FPGA zu laden.

6.3.1.1 Lattice ispVM und JTAG-Download-Kabel

Während des Entwicklungsprozesses des FPGA-Designs steht üblicherweise nur die Kon guration über ein spezielles JTAG-Download-Kabel zur Verfügung. Sie basiert auf der Verwendung der sogenannten JTAG-Schnittstelle des FPGA-Bausteins. Diese Schnittstelle ist für vielfältige Test- und Programmieraufgaben geeignet. Alle modernen FGPA-Bausteine verfügen über eine solche Schnittstelle, über die sie kon guriert werden können.

Lieferant der Kon gurationsdaten ist ein PC. Das hierzu notwendige Programm namens ispVM System kann kostenlos von der WWW-Page der Firma Lattice (`http://www.latticesemi.com` – Downloads – Software – Top Downloads: ispVM) geladen werden. Hierzu ist eine Registrierung auf der Lattice-Homepage erforderlich. Der Installer von ispVM System hat eine Dateigröße von ca. 25 MByte.

Zum Anschluss des PCs an die JTAG-Schnittstelle des FPGAs ist ein sogenanntes Programierkabel notwendig. Dieses Kabel ist in zwei Ausführungen erhältlich, nämlich für die USB-Schnittstelle

(Preis: ca. 150 €) und für den Parallel-Port (Preis: ca. 65 €). Genaueres ndet sich auf der Lattice-Homepage unter den Links „*Products – Development Hardware – Programming Cables*“. Auf der SandboxS 2 ist der JTAG-Port des FPGAs über die 10-polige Steckwanne mit der Bezeichnung U4

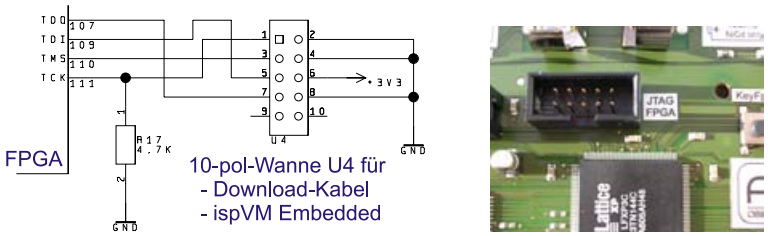


Abbildung 6.2: Steckwanne zum Anschluss an die JTAG-Schnittstelle des FPGAs.

(Abb. 6.2) erreichbar.

Verkauft werden die Produkte von Lattice über Distributoren. Deren aktuelle Adressen lassen sich ebenfalls über die oben genannte Homepage (Menüpunkt „*Sales*“) heraus nden. Die FH-OÖ/Hagenberg steht in keinem kommerziellen Verhältnis zu Lattice oder seinen Distributoren und verkauft natürlich auch keine Produkte von Lattice (oder anderen Herstellern). Bei Interesse an einem der genannten Download-Kabel wenden Sie sich also bitte direkt an den zuständigen Distributor.

Alternativ zur käuflichen Lösung kann ein Programmierkabel auch im Selbstbau erstellt werden. Zumindest für die Version zum Anschluss an den Parallelport ist dies relativ leicht möglich.

Installation von ispVM System

Die Installation geht wie bei den meisten anderen Windows-Programmen von statten. Als Installationsoption im Fenster mit dem Titel „*Select Components*“ sollte sicherheitshalber „*Full Installation*“ ausgewählt werden.

Im Fenster „*Select Additional Tasks*“ kann man getrost die Voreinstellungen übernehmen. Wer seinen Desktop frei von weiteren Icons halten möchte (in diesem Fall vier davon), kann natürlich das Häkchen im entsprechenden Auswahlfeld löschen.

Nach dem Infofenster mit einem Überblick der ausgewählten Komponenten öffnet ein Klick auf „*Install*“ ein Fenster mit der Frage, ob auch ein „*LSC USB or Parallel Port driver*“ installiert werden soll. Hier unbedingt mit „Ja“ antworten.

Abbildung 6.3: Auswahlfenster für die Treiberinstallation zum Download-Kabel von Lattice.

Nun beginnt das Kopieren der Dateien, das kurze Zeit darauf mit einem weiteren Infofenster unterbrochen wird. Nach einem Klick auf „*Next*“ erscheint ein Auswahlfenster wie in Abb. 6.3. Nachdem die Optionen wie in der Abbildung erkennbar umgestellt wurden, wird der Driver mit Klick auf „*Install*“ installiert. Natürlich fragt der Installer nochmals nach, ob man sich auch ganz sicher ist. Mit Klick auf „*Close*“ wird das Fenster wieder geschlossen und der Installer fragt nach, ob er den Rechner nun neu starten darf um die Installation abzuschließen. Bevor ispVM System verwendet wird, sollte man diesen Neustart unbedingt durchführen.

Download der Konfiguration mit ispVM System

Die nachfolgende Beschreibung wurde für ispVM System in der Version 16.4.1 erstellt. Die Versionsnummern dieses Programms schreiten rasch voran, so dass es unmöglich ist, bei jeder neuen Version auch die Beschreibung zu renovieren. Sollte die Beschreibung nicht mehr zur aktuellen Version passen, sind wir für einen Hinweis dankbar (`http://forum.hexapod.at`).

Nach Starten von ispVM System ist das Hauptfenster (Abb. 6.4 zu sehen.

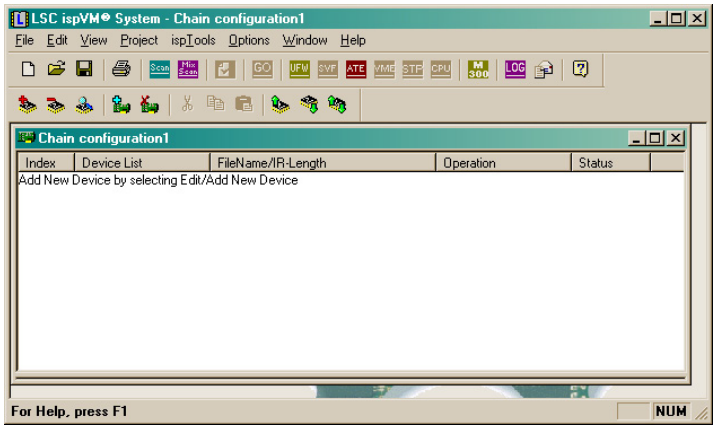


Abbildung 6.4: Hauptfenster von ispVM unmittelbar nach Start des Programmes.

Als zweiter Schritt folgt der Anschluss des Programmierkabels and den die SandboxS 2 (zehnpolige FPGA-JTAG-Anschlusswanne aus Abb. 6.2). Dann: Einschalten der SandboxS 2.

Nun wird in ispVM der Menüpunkt „*ispTools – Scan Chain*“ ausgewählt. Die Software sucht nun im System, dass über das Downloadkabel angeschlossen ist, nach Bausteinen, die in der sogenannten *Scan-Chain* miteinander und mit ispVM System verbunden sind. Das Ergebnis dieses Vorgangs ist in Abb. 6.5 zu sehen. In der *Scan-Chain* wurde ein Baustein des Typs LFXP3 aufgefunden. Damit ist das FPGA auf der SandboxS 2 gemeint.

Ein Doppelklick auf die Zeile mit dem Device LFXP3C wie in Abbildung 6.6 zu sehen,) öffnet das Fenster links in Abb. 6.7. In diesem Fenster wird der Bezug zur FPGA-Kon gurationsdatei hergestellt. Hierzu mit dem Button „*Browse*“ beispielsweise die Datei `hsserobchipv2.jed` suchen. Dann den Button „*OK*“ klicken. Es erscheint ein Infofenster (Abb. 6.7, rechts), das quittiert wird.

Die Datei kann nun mit „*Project – Download*“ aus dem Menü in das FPGA geladen werden. Dort wird sie in einem Flash-Speicher abgelegt und steht damit auch beim nächsten Einschalten der Spannungsversorgung im FPGA zur Verfügung. Während des Download-Vorgangs von 20-40 Sekunden Dauer erscheint das Fenster links in Abb. 6.8. Der erfolgreiche Abschluss des Downloads ist am Status „*PASS*“ (mit grünem Punkt) erkennbar. Im rechts in Abb. 6.8 gezeigten Fenster ist dies zu sehen.

Falls es zu Problemen kommt, ist das Downloadkabel oder der Parallel-Port die häu gste Ursache. Ob das Kabel von ispVM System erkannt wird, kann mit (Menü) „*Options – Cable and I/O Port Setup...*“ geprüft werden. Es öffnet sich ein Fenster (siehe Abb. 6.9) in dem mittels des Buttons „*Auto*

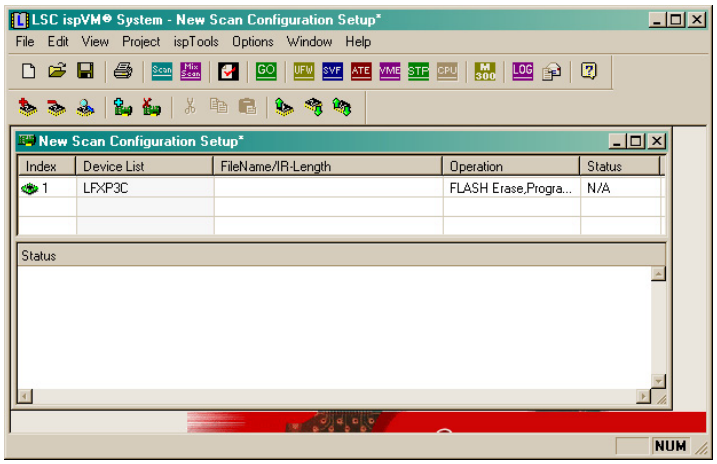


Abbildung 6.5: Die egScan-Chain wurde von ispVM erkannt. Der Baustein LFXP3C ist das FPGA auf der SandboxS 2. Es ist der einzige Baustein in der *Scan-Chain*.

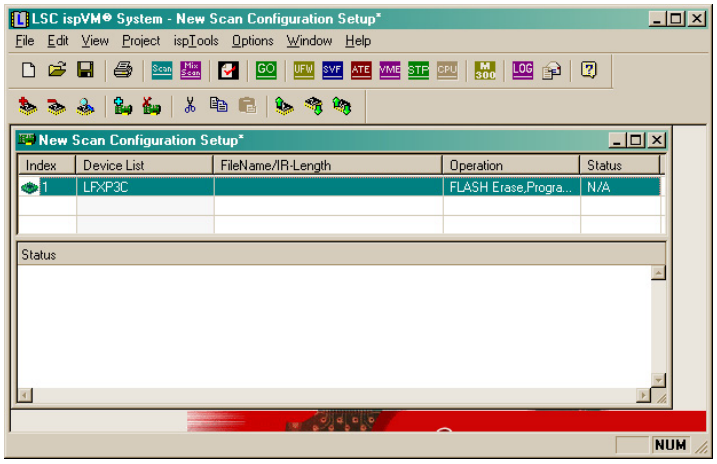


Abbildung 6.6: Hauptfenster von ispVM unmittelbar nach Start des Programmes.

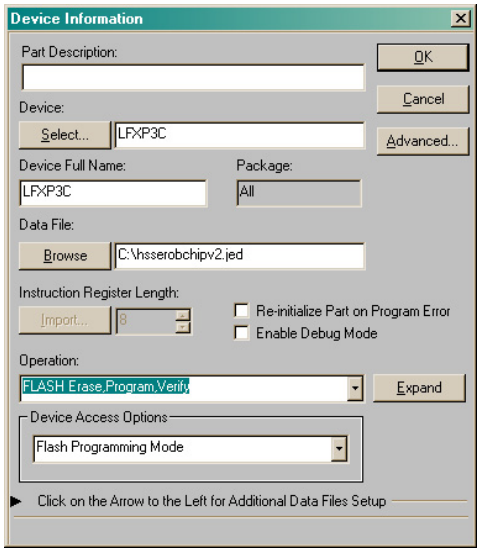


Abbildung 6.7: Dialogfenster zum Eintragen der Kon gurationsdatei und Infofenster als Reaktion auf das Schließen des Dialogfensters.

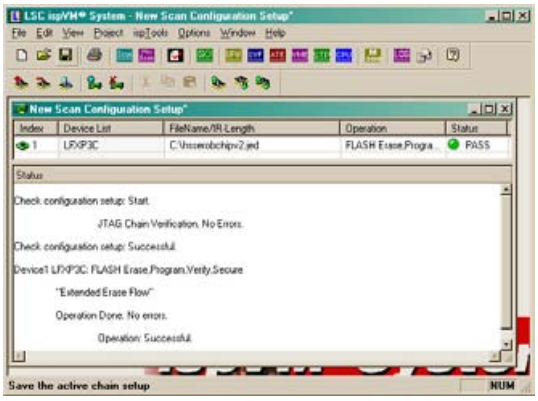


Abbildung 6.8: Der Downloadvorgang dauert ca. 20 bis 40 Sekunden und schließt (hoffentlich) korrekt, also mit dem Kennzeichen „*PASS*“ ab.

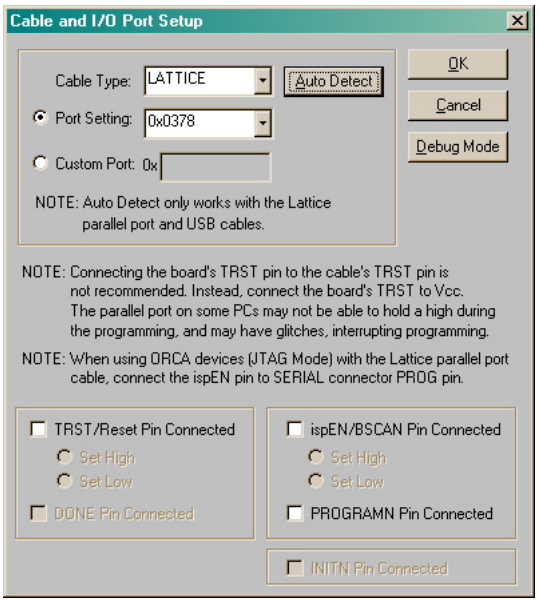

Cable and I/O Port Setup
Cable Type: LATTICE
Auto Detect
Port Setting: 0x0378
Custom Port: 0x
NOTE: Auto Detect only works with the Lattice parallel port and USB cables.
OK
Cancel
Debug Mode
NOTE: Connecting the board's TRST pin to the cable's TRST pin is not recommended. Instead, connect the board's TRST to Vcc. The parallel port on some PCs may not be able to hold a high during the programming, and may have glitches, interrupting programming.
NOTE: When using ORCA devices (JTAG Mode) with the Lattice parallel port cable, connect the ispEN pin to SERIAL connector PROG pin.
TRST/Reset Pin Connected
Set High
Set Low
DONE Pin Connected
ispEN/BSCAN Pin Connected
Set High
Set Low
PROGRAMN Pin Connected
INITN Pin Connected

Abbildung 6.9: Kabeltest in ispVM System.

Detect“ das Kabel gesucht werden kann. Ein Test mit abgezogenem Kabel zeigt, dass das fehlende Kabel mit einer entsprechenden Fehlermeldung quittiert wird. Ist das Kabel angesteckt, passiert bei Druck auf den Button dagegen (scheinbar) nichts.

Daneben gibt es übrigens eine noch häu gere Fehlerquelle: Es wurde vergessen, die SandboxS 2 einzuschalten oder deren Akkuzellen sind entladen.

6.3.1.2 Embedded Programming: Lattice ispVM Embedded und AVR

Die JTAG-Schnittstelle des FPGAs arbeitet mit Logikpegeln und nach einem speziellen seriellen Protokoll. Es ist möglich, dieses Protokoll in Software nachzubilden und die Programmierung des Flash-Speichers im FPGA von einem Mikrocontroller aus vorzunehmen. Hierzu ist allerdings eine Verbindung zwischen dem Mikrocontroller und dem JTAG-Port des FPGAs notwendig. Da dieser Port entsprechend des vorangegangenen Abschnitts auch an ein Download-Kabel angeschlossen werden soll, wäre ein (teurer) Umschalter zwischen der Kabelverbindung und der Verbindung zum Mikrocontroller notwendig. Aus Kostengründen wurde hierauf verzichtet, da alternativ auch eine Kabelverbindung über den JTAG-Port-Stecker zum Mikrocontroller hergestellt werden kann.

Lattice stellt unter dem Namen ispVM-Embedded das Gerüst für einen sogenannten JTAG-Sequencer zur Verfügung. Diese Sammlung von C-Programmen kann an die Bedürfnisse der eigenen Anwendung angepasst werden. Konkret bedeutet dies, dass Anpassungen für den eingesetzten C-Compiler und vor allem für den verwendeten Mikrocontroller und dessen Schnittstellen gemacht werden müssen. Endergebnis dieser Modi kationen ist eine Firmware für einen Mikrocontroller, welche die Programmierung in der gleichen Weise erledigt, wie das PC-basierte Programm ispVM System, das im vorigen Abschnitt besprochen wurde.

Diese Art der Programmierung wurde bislang noch nicht für die SandboxS 2 umgesetzt, da ein Programmierkabel in Verbindung mit ispVM System nur unwesentlich teurer, aber ungleich flexibler verwendbar ist.

6.3.1.3 Flash Direct Programming: Lattice CPU Embedded und AVR

Alternativ zur Programmierung des FPGAs über die JTAG-Schnittstelle steht der sogenannte ispCon g-Port des FPGAs zur Verfügung. Auch hierfür stellt Lattice ein Programmgerüst für einen Mikrocontroller zur Verfügung, das sich CPU-Embedded nennt. Ein Vorteil dieser Lösung liegt im erheblich kleineren Aufwand für die Mikrocontrollersoftware. Die Anpassung des Gerüstes an die konkreten Verhältnisse erfordert ähnliche Schritte wie bei der im vorigen Abschnitt besprochenen Lösung.

Das Design der SandboxS 2 sieht bereits alle notwendigen Verbindungen zwischen AVR und FPGA für diese Art der Programmierung vor. Die entsprechenden Leitungen werden doppelt genutzt: Für das External Memory Interface des AVR-Mikrocontrollers und zugleich für die Programmierung des FPGAs über die ispCon g-Schnittstelle. Der JTAG-Anschluss wird hierdurch nicht belegt.

Das *Flash Direct Programming* des Flash-Kon gurationsspeichers im FPGA wird derzeit auf Grund eines Problems im FPGA-Chip von Lattice (noch) nicht unterstützt. Laut Lattice wird eine Mindestfrequenz des Programmiertaktes von 200 kHz benötigt. Bei Verwendung des vorgegebenen Programmgerüstes ist diese Frequenz aber mittels eines AVR-Mikrocontrollers nicht erreichbar.

7 Robotik-Peripherie: HSSE-Robotics-Chip

Der HSSE-Robotics-Chip V2 stellt dem Mikrocontroller und damit auf Softwareebene Peripheriekomponenten (Abb. 7.1) zur Verfügung, die speziell auf die Anwendung in der Robotik zugeschnitten

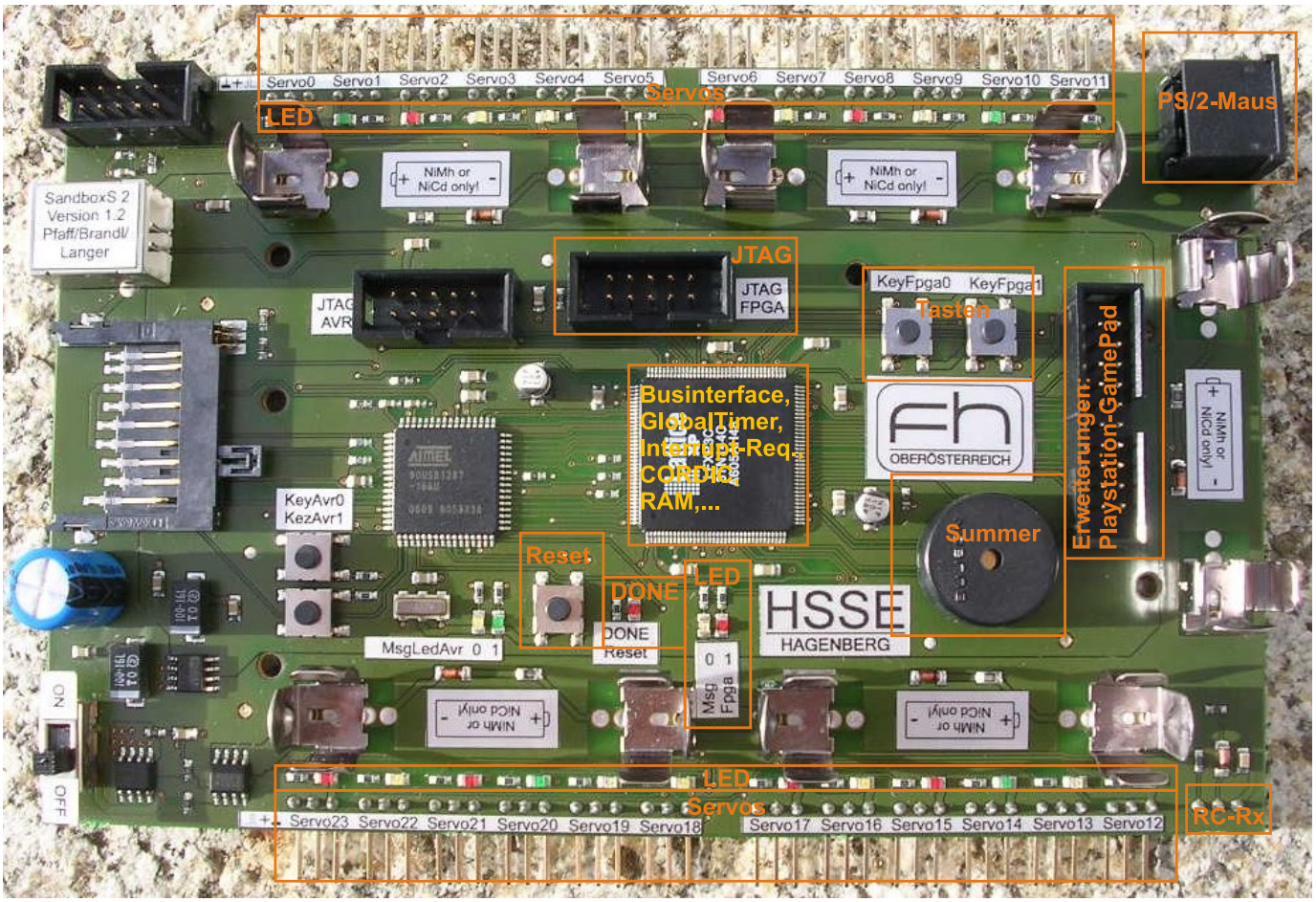


Abbildung 7.1: Das FPGA vom Typ Lattice XP stellt spezielle Peripherieeinheiten für die Robotik und ähnliche Anwendungsbereiche zur Verfügung. Die zugehörigen Bereiche auf der Platine sind hervorgehoben.

sind:

- Erzeugen von PWM-Signalen zur Ansteuerung von 24 handelsüblichen Modellbauservos,
- Berücksichtigung von Korrekturwerten bei der Umrechnung von Sollwinkel auf den PWM-Wert zur Servosteuerung und der
- Anpassung der Bewegungsgeschwindigkeit der Servos, so dass alle Servos sich synchron und

gleichförmig bewegen, wobei die Sollwinkel in einem festen Zeitraster erreicht werden. Zum entsprechenden Zeitpunkt erfolgt die

- Ausgabe eines Interrupt-Signals an den Mikrocontroller um die Bereitschaft für neue Zielwinkel aller Servos mitzuteilen,
- die Mittelstellung der Servos kann zur Justage und als Servotest per Tastendruck angewählt werden,
- CORDIC-Coprozessor zur Berechnung der inversen Kinematik und zur Koordinatentransformation,
- Einlesen des PPM-Signals aus der Demodulationsstufe eines Modellbauempfängers,
- Einlesen der Tasten- und Joystickstellungen von einem handelsüblichen kabelgebundenen oder drahtlosen Playstation-2-GamePad
- Einlesen der Bewegungsdaten einer PS/2-Maus und
- feinstu ge Helligkeitssteuerung von 24 LEDs.

Diese Funktionen wurden auf der Abstraktionsebene RTL (*Register Transfer Level*) in VHDL modelliert, wobei hauptsächlich das Entwurfsparadigma der hierarchischen FSMD verwendet wurde. Zur Simulation wurde Mentor Graphics Modelsim eingesetzt. Synthesewerkzeug war Synplicity Synplify in der Lattice-Version. Als Backend-Werkzeug kam die Toolsuite ispLever von Lattice in der Starter-Version zum Einsatz.

Der Themenbereich Chip-Design ist fester Bestandteil der Studieninhalte des Studiengangs HSSE an der FH-OÖ/Hagenberg. Einfache Designs werden bereits im 3. Semester entworfen und in der Praxis erprobt. Die Methoden, die für den Entwurf des HSSE-Robotics-Chip benötigt werden, vermittelt das Studium im 4. Semester. Im 5. Semester rundet das Design einer 16-Bit-CPU und eines Signalverarbeitungssystems (Audio-/Kommunikationstechnik) das Thema für die Bachelor-Ausbildung ab.

Der aufbauende Master-Studiengang *Embedded Systems Design* (ESD) führt diesen Ausbildungszweig mit den zukunftsweisenden Themen System-On-Chip-Design und Advanced Veri cation weiter.

7.1 Peripheriekomponenten aus Sicht des Mikrocontrollers

Die nachfolgenden Abschnitte zeigen die Fähigkeiten der einzelnen Peripheriekomponenten auf, welche das FPGA dem Mikrocontroller zur Verfügung stellt. Diese Darstellung ähnelt den Erläuterungen der Peripheriekomponenten aus dem Datenblatt des AT90USB1286/7 (Timer, SPI, USART, ADC,...).

Die Gesamtheit aus Atmel AVR und FPGA erscheint der Software tatsächlich wie ein AVR, der Robotik-Peripheriekomponenten besitzt. Aus Sicht der Software be ndet man sich also nach Studium der Dokumentation dieser Peripheriekomponenten nach wie vor auf gewohntem Terrain. Leider gibt es jedoch eine Einschränkung: Der AVR-Simulator und der Debugger JTAG ICE (II) wissen von den neu hinzugewonnenen Fähigkeiten wenig bis nichts. Hierzu müsste man die Robotik-Peripherie auch in deren Entwicklungssoftware nachbilden.

In Hinsicht auf das Echtzeitverhalten der Robotik-Peripheriekomponenten ergibt sich leider ein weiteres Problem des Softwaredebugging. Ähnlich wie bei den Timern und beim ADC des AVR läuft

auch die Servosteuerung unabhängig vom Debuggingzustand der AVR-CPU. Dieses Verhalten stellt ein typisches Problem des *Debugging* von Echtzeitanwendungen dar und lässt sich mit den typischerweise zur Verfügung stehenden Mitteln nicht ohne weiteres lösen. Hierzu sind spezielle Debugging-Werkzeuge notwendig.

7.1.1 Zugriff auf 16-Bit-Register

Lese- und Schreibzugriffe des Atmel AVRs erfolgen stets auf 8 Bit breite Datenwörter. Der Zugriff auf 16 Bit breite Register muss daher durch zwei Zugriffe mit 8 Bit Datenwortbreite statt nden. Beim Lesezugriff besteht die Gefahr, dass sich die Registerwerte zwischen den beiden Leseoperationen ändern, so dass nicht zusammengehörende 8 Bit breite Hälften eines Wortes gelesen werden. Genauso lässt der auf zwei Operationen aufgeteilte Schreibzugriff das Register in der Zeit zwischen den beiden Zugriffen in einem ungültigen Status.

Dieses Problem tritt auch bei einigen Peripherieeinheiten des AVRs auf. So beispielsweise beim 16-Bit-Timer. Im Datenblatt ndet sich hierzu eine genaue Beschreibung. Darüber hinaus existiert eine *Application Note* mit der Bezeichnung AVR072, die detailliert auf dieses Problem eingeht. Diese *Application Note* steht auf der WWW-Seite von Atmel (`http://www.atmel.com/dyn/resources/prod_documents/DOC1493.PDF`) zur Verfügung. Die Lösungsstrategie für dieses Problem lässt sich aus Sicht der Software folgendermaßen zusammenfassen:

> *To do a 16-bit write, the high byte must be written before the low byte. For a 16-bit read, the low byte must be read before the high byte.*

Bei einem 16-Bit-*Schreib*zugriff wird also zuerst das höherwertige Byte (das Byte mit der höheren Adresse) geschrieben, dann erst das niederwertige Byte, also jenes mit der niedrigeren Adresse. Umgekehrt wird bei einem *Lese*zugriff das niederwertige Byte (jenes mit der niedrigeren Adresse) vor dem höherwertigen Byte (dem mit der höheren Adresse) gelesen.

Zwischen den beiden Zugriffen auf *Low Byte* und *High Byte* darf keine Interrupt-Verarbeitung erfolgen. Im Zuge solcher Interrupt-Routinen könnte das temporäre Register zur Speicherung des *High Byte* verändert werden. In Folge würde beim Schreiben oder Lesen des *Low Byte* dieses veränderte *High Bytes* übernommen werden. Daher sollte vor Zugriffen auf 16 Bit breite Register das Interrupt-Flag des Prozessors gesetzt werden, um Interrupts gänzlich zu unterbinden. Nachdem beide 8-Bit-Zugriffe fertiggestellt sind, können Interrupts wieder freigegeben werden.

Das *Low Byte* steht an einer geraden Adresse (A[0] hat den Wert `'0'`), das zugehörige *High Byte* be ndet sich an der folgenden Adresse, die also stets ungerade ist (A[0] = `'1'`).

Die meisten Parameterregister des HSSE-Robotics-Chips sind 16 Bit breit, so dass für einen Zugriff auf das komplette Register die oben angesprochene Reihenfolge einzuhalten ist. Soll lediglich auf die unteren 8 Bit eines Registers zugegriffen werden oder sind ohnehin nur diese 8 Bit belegt, so kann der Zugriff unmittelbar in einem AVR-Schreib- oder -Lesezyklus geschehen.

7.2 Interrupts

Der HSSE-Robotics-Chip kann dem Mikrocontroller das Eintreten verschiedener Ereignisse mittels der External-Interrupt-Pins INT0 und INT1 mitteilen:

INT1 Zuletzt durch den Mikrocontroller aufgegebener Servosequenzauftrag bestehend aus PhiStart, DeltaPhi und NrOfDeltas wird nun bearbeitet. Daher kann ein neuer Auftrag angenommen wer-

den. Dieses Flag existiert einzeln für jedes Servo. Die Auftragsannahme geschieht intern in der Reihenfolge der Servonummern, so dass es meist ausreicht, das Servo mit der höchsten Nummer zur Auslösung dieses Interrupts zu verwenden. Wenn alle Servos die gleiche Anzahl an Delta-Schritten vorgegeben wird, können so alle Servos im gleichen Arbeitstakt mit Bewegungssequenzen versorgt werden.

INT0 Neue Daten sind von der PS/2-Schnittstelle eingetroffen. Da diese Daten für die angeschlossene Maus relativ zur zuletzt eingenommenen Position angegeben werden, darf dieses Ereignis nicht übersehen werden.

INT0 Neue Daten vom RC-Empfänger liegen bereit. Dies bedeutet nicht, dass sich die Daten seit dem letzten Auslesen geändert haben müssen.

INT0 Servogrundstellungsmodus wurde eingeschaltet.

INT0 Paniktaste wurde gedrückt.

INT0 CORDIC-Ergebnisse liegen vor.

All diese Ereignisse können einzeln zur der Interrupt-Auslösung zugelassen werden, indem das entsprechende Interrupt-Mask-Bit (IrqMask) gesetzt wird (Wert 1). Die Interrupt-Mask-Bits nden sich in den in den Abschnitten 7.5.3.1, 7.13.1.1 und 7.22 beschrieben.

Welche Quelle den Interrupt ausgelöst hat, lässt sich durch Lesen der Interrupt-Flag-Register feststellen. Die Interrupt-Flag-Register sind 8 Bit breit, so dass kein 16-Bit-Zugriff laut Abschnitt 7.1.1 durchgeführt werden braucht. Stattdessen werden die Interrupt-Flags immer in Portionen zu einem Byte gelesen.

Auslesen des jeweiligen Registers löscht zugleich alle im Register gesetzten Flags. Die Daten werden sozusagen nicht nur betrachtet, sondern entnommen.

Es müssen also sämtliche Interrupt-Requests, welche aus einem Interrupt-Flag-Register ausgelesen wurden, nacheinander verarbeitet werden. Sollte zufällig gleichzeitig mit dem Auslesen eines der Interrupt-Flag-Register ein neuer Interrupt eintreffen, so wird dieser im Register abgelegt und alle zuvor bereits vorhandenen Interrupt-Flags werden gelöscht.

Der Interrupt-Ausgang bleibt nach dem Aktivieren durch die Interrupt-Quelle solange aktiv, bis alle ihm zugeordneten Interrupt-Flag-Register (mit aktiviertem Interrupt-Mask-Bit) durch Lesen geleert wurden. Dieses Verhalten ermöglicht auf Wunsch die Verwendung des *Level-Controlled-Interrupt* des AVR.

Es ergibt sich folgende Vorgehensweise zur Verarbeitung der Interrupt-Anforderungen, die vom HSSE-Robotics-Chip kommen:

- External-Interrupts von den Pins INT0 und INT1 des AVR-Mikrocontrollers während der Initialisierungsphase freigeben.
- Interrupt-Mask-Bits der Peripherieeinheiten setzen (Wert 1), wenn diese einen Interrupt-Request auslösen sollen. Häu gster Fall ist die Freigabe von Interrupt-Requests für den Servosequencer des Servos mit der höchsten Servonummer.
- Interrupt-Routinen für die beiden Interrupt-Anforderungen von INT0 und INT1 erstellen. Grundsätzlicher Aufbau beider Routinen:

a) Auslesen und Zwischenspeichern *aller* zum auslösenden Interrupt-Request gehörenden Interrupt-Flag-Register. Hierdurch wird die Interrupt-Anforderung seitens des Robotics-Chips zurückgenommen und es werden alle ausgelesenen Interrupt-Flags gelöscht.

b) Bearbeiten aller maßgeblichen Interrupt-Flags aus allen ausgelesenen Interrupt-Flag-Registern.

c) Rückkehr aus der Interrupt-Service-Routine des External-Interrupt-Requests (INT0, INT1).

Alternativ kann statt einer Interrupt-Verarbeitung auch ein *Polling* auf die Interrupt-Flag-Register durchgeführt werden. Hierzu werden die entsprechenden (oder alle) Interrupt-Mask-Bits deaktiviert (Wert 0), damit kein Interrupt-Request ausgelöst wird. Um festzustellen, ob eine Interrupt-Anforderung vorliegt, werden alle Interrupt-Flag-Register beständig zyklisch ausgelesen und die darin enthaltenen Interrupt-Flags ausgewertet.

7.2.1 Interrupt-Flag-Register

Adr(AVR)	Interrupt-Flag-Register
0x9C00	Bereit für neue Servo-Bewegungssequenz, Servos 0 – 7
0x9C01	Bereit für neue Servo-Bewegungssequenz, Servos 8 – 15
0x9C02	Bereit für neue Servo-Bewegungssequenz, Servos 16 – 23
0x9C03	IRQ-Flags für PS/2, RC-Rx, Grundstellung, Paniktaste und CORDIC

Tabelle 7.1: Addressraum der Interrupt-Flag-Register. Diese Register haben eine Datenwortbreite von 8 Bit, so dass der Mikrocontroller das gesamte Register mit einem einfachen Zugriff lesen kann. Die Register sind nur les-, aber nicht schreibbar.

7.2.1.1 Register ServoSeqEmpty70

Bit	7	6	5	4	3	2	1	0
0x9C00	**IFS7**	**IFS6**	**IFS5**	**IFS4**	**IFS3**	**IFS2**	**IFS1**	**IFS0**

Tabelle 7.2: Register: ServoSeqEmpty70. Im Initialisierungszustand nach dem Einschalten und nach Reset sind alle Interrupt-Flags inaktiv (Wert 0). Nur lesbar. Durch das Auslesen des Registers werden alle Interrupt-Flags im Register gelöscht.

Bit 0 — IFS0 Interrupt-Flag Servosequencer 0. Dieses Flag wird gesetzt (Wert 1), wenn der zuvor vom Mikrocontroller geschriebene Servosequenzauftrag (bestehend aus den Werten für Phi-Start, DeltaPhi und NrOfDeltas) vom Servosequencer abgeholt wurde. Ist das Interrupt-Mask-Bit ImSqe im Register ModeIrqMask von Servo 0 gesetzt, wird bei gesetztem Interrupt-Flag ein Interrupt auf der Leitung INT1 beim Mikrocontroller angefordert.

Bits 1 – 7 — IFS1 – IFS7 Äquivalent zum Interrupt-Flag für den Servosequencer 0 gibt es die entsprechenden Interrupt-Flags auch für die Servosequencer 1 bis 7.

7.2.1.2 Register ServoSeqEmpty158

Bit	7	6	5	4	3	2	1	0
0x9C01	**IFS15**	**IFS14**	**IFS13**	**IFS12**	**IFS11**	**IFS10**	**IFS9**	**IFS8**

Tabelle 7.3: Register: ServoSeqEmpty158. Im Initialisierungszustand nach dem Einschalten und nach Reset sind alle Interrupt-Flags inaktiv (Wert 0). Nur lesbar. Durch das Auslesen des Registers werden alle Interrupt-Flags in diesem Register gelöscht.

Bits 0 – 7 — IFS8 – IFS15 Äquivalent zum Interrupt-Flag für den Servosequencer 0 gibt es die entsprechenden Interrupt-Flags auch für die Servosequencer 8 bis 15.

7.2.1.3 Register ServoSeqEmpty2316

Bit	7	6	5	4	3	2	1	0
0x9C02	**IFS23**	**IFS22**	**IFS21**	**IF20**	**IFS19**	**IFS18**	**IFS17**	**IFS16**

Tabelle 7.4: Register: ServoSeqEmpty2316. Im Initialisierungszustand nach dem Einschalten und nach Reset sind alle Interrupt-Flags inaktiv (Wert 0). Nur lesbar. Durch das Auslesen des Registers werden alle Interrupt-Flags im Register gelöscht.

Bits 0 – 7 — IFS16 – IFS23 Äquivalent zum Interrupt-Flag für den Servosequencer 0 gibt es die entsprechenden Interrupt-Flags auch für die Servosequencer 16 bis 23.

7.2.1.4 Register IrqFlagsOthers

Bit	7	6	5	4	3	2	1	0
0x9C03	—	—	—	**IFCord**	**IFNeut**	**IFPnk**	**IFRc**	**IFPs2**

Tabelle 7.5: Register: IrqFlagsOthers. Im Initialisierungszustand nach dem Einschalten und nach Reset sind alle Interrupt-Flags inaktiv (Wert 0). Nur lesbar. Durch das Auslesen des Registers werden alle Interrupt-Flags im Register gelöscht.

Bit 0 — IFPs2 Interrupt-Flag zur Anzeige der Verfügbarkeit neuer Daten vom PS/2-Mausinterface (Abschnitt 7.14). Ist das zugehörige Mask-Bit ImPs2 im Register IrqMaskRcRxCtrl (Abschnitt 7.13.1.1) gesetzt, wird auch ein Interrupt auf der Leitung INT0 des Mikrocontrollers angefordert.

Bit 1 — IFRc Interrupt-Flag zur Anzeige der Verfügbarkeit neuer Daten vom RC-Empfängerdecoder (Abschnitt 7.13). Ist das zugehörige Mask-Bit ImRc im Register IrqMaskRcRxCtrl (Abschnitt 7.13.1.1) gesetzt, wird auch ein Interrupt auf der Leitung INT0 des Mikrocontrollers angefordert.

Bit 2 — IFPnk Dieses Interrupt-Flag wird bei Druck auf die Servopaniktaste (Abschnitt 7.9) gesetzt. Auslöser für das Setzen des Flags ist der erstmalige Druck auf die Taste. Auch wenn das Flag

durch Auslesen gelöscht wird, kann es kein zweites Mal gesetzt werden. Ist das zugehörige Mask-Bit ImPnk im Register IrqMaskRcRxCtrl (Abschnitt 7.13.1.1) gesetzt, wird auch ein Interrupt auf der Leitung INT0 des Mikrocontrollers angefordert.

Bit 3 — IFNeut Dieses Interrupt-Flag wird aktiv, sobald der Servogrundstellungsmodus (Abschnitt 7.8) gewählt wird. Auslösend wirkt sich der Tastendruck zum Wechsel in den Servogrundstellungsmodus aus. Dieser setzt das Interrupt-Flag. Ist das zugehörige Mask-Bit ImNeut im Register IrqMaskRcRxCtrl (Abschnitt 7.13.1.1) gesetzt, wird auch ein Interrupt auf der Leitung INT0 des Mikrocontrollers angefordert.

Bit 4 — IFCord Dieses Interrupt-Flag wird aktiv, sobald der CORDIC-Prozessor (Abschnitt 7.22) einen Berechnungsauftrag vollständig bearbeitet hat. Ist das zugehörige Mask-Bit ImCord im Register IrqMaskRcRxCtrl (Abschnitt 7.13.1.1) gesetzt, wird auch ein Interrupt auf der Leitung INT0 des Mikrocontrollers angefordert.

7.3 Servosteuerung

Modellbauservos und spezielle Robotikservos erfreuen sich im Bereich der *Embedded Robotics* hoher Beliebtheit. Hierfür sprechen einige gute Gründe:

- Sie sind leicht und zu einem relativ geringen Preis erhältlich,
- es gibt zahlreiche Bauformen und
- ihre Ansteuerung ist weitgehend genormt.

Abbildung 7.2: Beispiele für Robotik- und Modellbauservos.

Servos erfüllen eine einfache Aufgabe: Man sendet auf elektrischem Wege den Befehl „Stelle den Servoarm (oder Servoscheibe) auf den Winkel 67°“ und das Servo erledigt dies mit der ihm eigenen Geschwindigkeit, Kraft und Genauigkeit.

Angetrieben wird der Servoarm üblicherweise von einem kleinen Gleichstrommotor, der den Servohebel über ein Getriebe verdrehen kann. Um den Motor mit elektrischer Energie zu versorgen, werden zumindest zwei Anschlüsse gebraucht: Masse (GND) und die positive Versorgungsspannung (typischerweise zwischen 4 und 6,5 V).

Der oben angesprochene Befehl für den Stellwinkel wird über ein drittes Kabel übertragen, das ebenfalls das Massekabel als Rückleitung verwendet. Die Mitteilung des gewünschten Winkels erfolgt

über die Dauer eines Spannungsimpulses (von mehr als 2 V). Der außerhalb dieser Zeitdauer anliegende Spannungspegel liegt nahe bei 0 V, so dass die beiden Spannungen leicht unterscheidbar sind. Die Dauer des Impulses bestimmt unmittelbar die Winkelstellung des Servoarms. In Abbildung 7.3 ist der Zusammenhang graphisch dargestellt. Der HSSE-Robotics-Chip erzeugt mittels eines vom Mi-

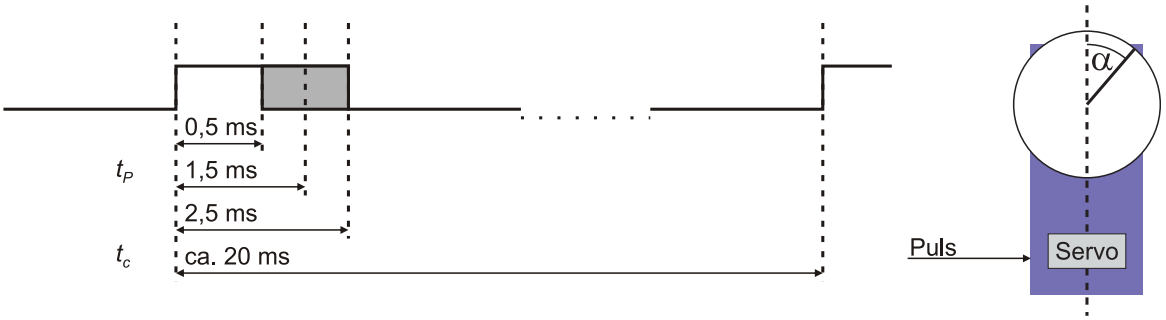


Abbildung 7.3: Die Pulsdauer steuert den Servostellwinkel.

krocontroller aus parametrisierbaren Pulsgenerators Impulse zur Steuerung von 24 Servos.

Abbildung 7.4 zeigt Beispiele von Impulslängen und zugehörigen Servohebelwinkeln. Das Ser-

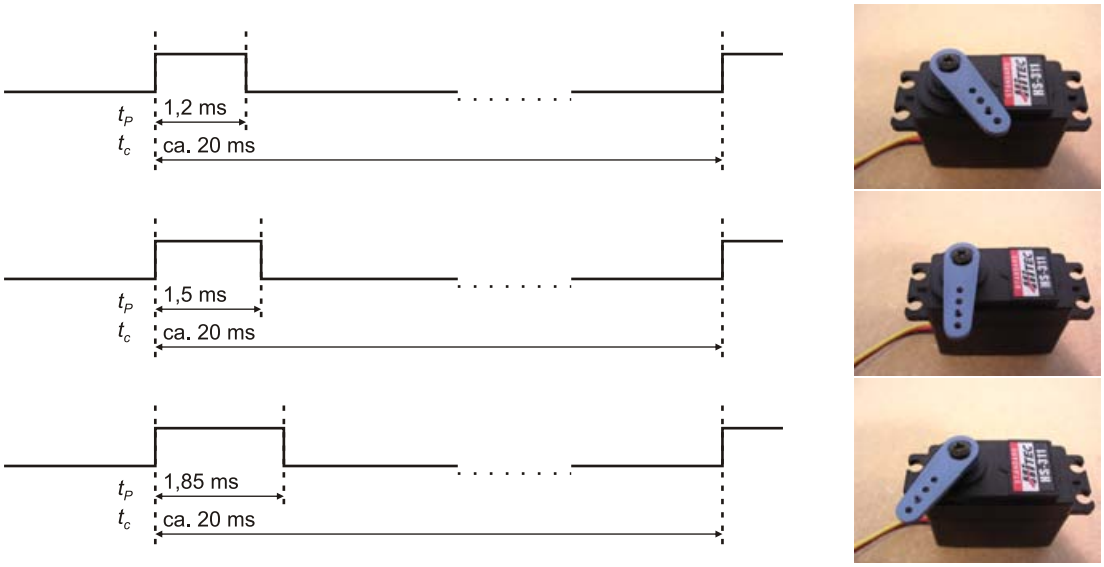


Abbildung 7.4: Beispiele für die Stellung des Servoarms in Abhängigkeit von der Impulsdauer.

vo im Bild stellt den Servohebel mit größer werdender Impulslänge im Uhrzeigersinn weiter. Die Richtung, in welche sich der Hebelarm bei größer werdender Impulslänge bewegt, ist jedoch nicht genormt. Selbst bei unterschiedlichen Servotypen des gleichen Hersteller kann die Drehrichtung einmal wie im Bild und das andere Mal umgekehrt eingestellt sein. Der HSSE-Robotics-Chip beseitigt dieses Problem durch eine eingebaute Drehrichtungsumkehr, die mittels Umrechnung der Impulsdauer durchgeführt wird.

Der Impuls muss regelmäßig wiederholt werden, was in der Praxis im Zyklus von 16 bis 30 ms geschieht. Wenn allzu lange kein Impuls beim Servo ankommt, entspannt das Servo sozusagen, d.h. der

Motor des Servos wird gänzlich abgeschaltet. Da kein Winkel gewünscht wird, ist nun jeder Winkel möglich und durch Verdrehen des Servohebels von außen auch tatsächlich einstellbar. Allerdings sollte man der Versuchung, am Hebel zu drehen, mit Rücksicht auf das Getriebespiel des Servos besser widerstehen. Der HSSE-Robotics-Chip bietet die Möglichkeit, die Impulserzeugung für jedes Servo zu unterdrücken. Hierdurch kann der Energieverbrauch eines Roboters unter Umständen erheblich reduziert werden. Nach dem Einschalten der Energieversorgung der SandboxS 2 sind alle Servoimpulse deaktiviert, damit die Akkuzellen durch den plötzlich einsetzenden Energiebedarf sämtlicher Servomotoren nicht überlastet werden. Sinnvollerweise wird die Impulserzeugung der einzelnen Servos nach dem Einschalten der SandboxS 2 nacheinander mit einigem Zeitabstand aktiviert, um die Akkuzellen und die Mechanik zu schonen. Geschieht dies nicht, ist es unter Umständen sogar möglich, dass die Akkuspannung auf Grund der Überlastung direkt nach dem Einschalten zusammenbricht und einen Unterspannungsreset des Mikrocontrollers und des FPGAs auslöst. Der Roboter lässt sich dann nicht in Betrieb nehmen, da der beschriebene Überlastungseffekt unmittelbar nach Verlassen des Resets sofort wieder auftritt. Erkennbar ist dieses Problem an der rhythmischen ruckartigen Bewegung sämtlicher Servos.

Einige Servos, insbesondere sogenannte Digitalservos, behalten den zuletzt eingestellten Winkel bei, statt den Motor abzuschalten. Solche Servos sind unter Umständen für den Einsatz mit der SandboxS 2 unbrauchbar. Allerdings kann man Digitalservos häu g auch so programmieren, dass sie bei fehlendem Impuls das im vorigen Absatz beschriebene Verhalten zeigen.

Funktionsweise eines Servos

Die Auswertung des Servosteuerimpulses (= Messung der Impulsdauer) und die Steuerung des Servomotors werden von der Servoelektronik durchgeführt. Diese Baugruppe muss auch den momentanen Ist-Winkel des Hebelarms kennen, damit sie darüber entscheiden kann, in welche Richtung der Hebel zum Erreichen des gewünschten Winkels zu verdrehen ist.

Der Ist-Winkel des Hebelarms wird hierzu von einem Drehpotentiometer (= einstellbarer Spannungsteiler) in einen Spannungswert übersetzt. Das Potentiometer dient also als Winkelsensor. Dieser Spannungswert wird von der Servoelektronik gemessen.

Damit schließt sich im Servo ein Regelkreis der in Abb. 7.5 zu sehen ist.

Modellbau- und Robotikservos werden in großen Stückzahlen unter enormem Preisdruck hergestellt. Verständlicherweise lassen sich konkurrenzfähige Preise nur erreichen, wenn die Fertigungstoleranzen der einzelnen Komponenten nicht unrealistisch eng gewählt werden. Daher ist die Genauigkeit der Zuordnung von Pulsdauer und damit dem Sollwinkel und dem tatsächlich resultierenden Servohebelwinkel stark von der Qualität der einzelnen Teile und ihrer Abstimmung aufeinander abhängig. Ungenauigkeiten schleichen sich insbesondere an folgenden Stellen in das System „Servo“ ein:

- Mechanisches Spiel zwischen Hebelarm und Potentiometerachse,
- Fertigungsgenauigkeit des Potentiometers,
- Abnutzung oder Alterung des Potentiometers,
- Messungenauigkeit bei der Bestimmung der Spannung, welche das Potentiometer vorgibt,
- Messungenauigkeit bei der Bestimmung der Impulsdauer (Sollwinkel),
- Nichtlineares Verhalten der Regelelektronik.

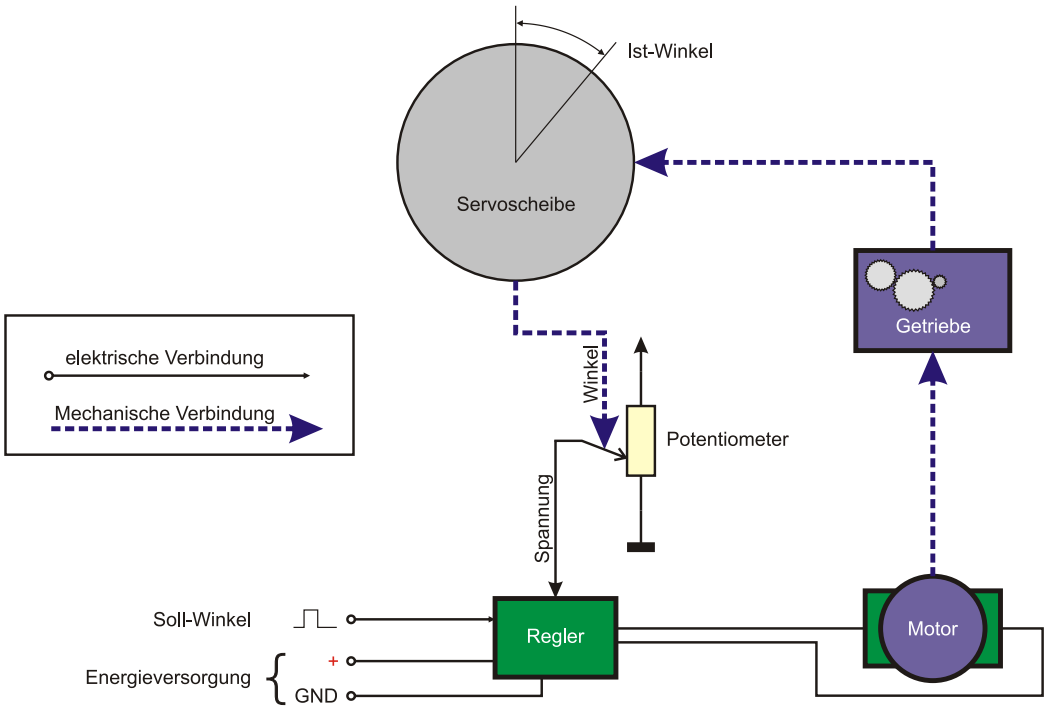


Abbildung 7.5: Regelkreis in einem Modellbauservo. Der Sollwinkel wird durch eine Impulsdauer von außen vorgegeben. Der elektronische Regler misst den Ist-Winkel des Hebelarms, wobei ein Potentiometer als Sensor fungiert und stellt den Servoarm mittels eines Elektromotors mit nachgeschaltetem Geriebe so ein, dass Soll- und Ist-Winkel im Rahmen einer bestimmten Toleranz übereinstimmen.

In Summe ergeben sich Ungenauigkeiten, die zwar im Modellbaubereich meist tolerabel sind, aber in der Robotik sehr störend wirken. In Abb. 7.6 ist der Zusammenhang von Pulslänge und einge-

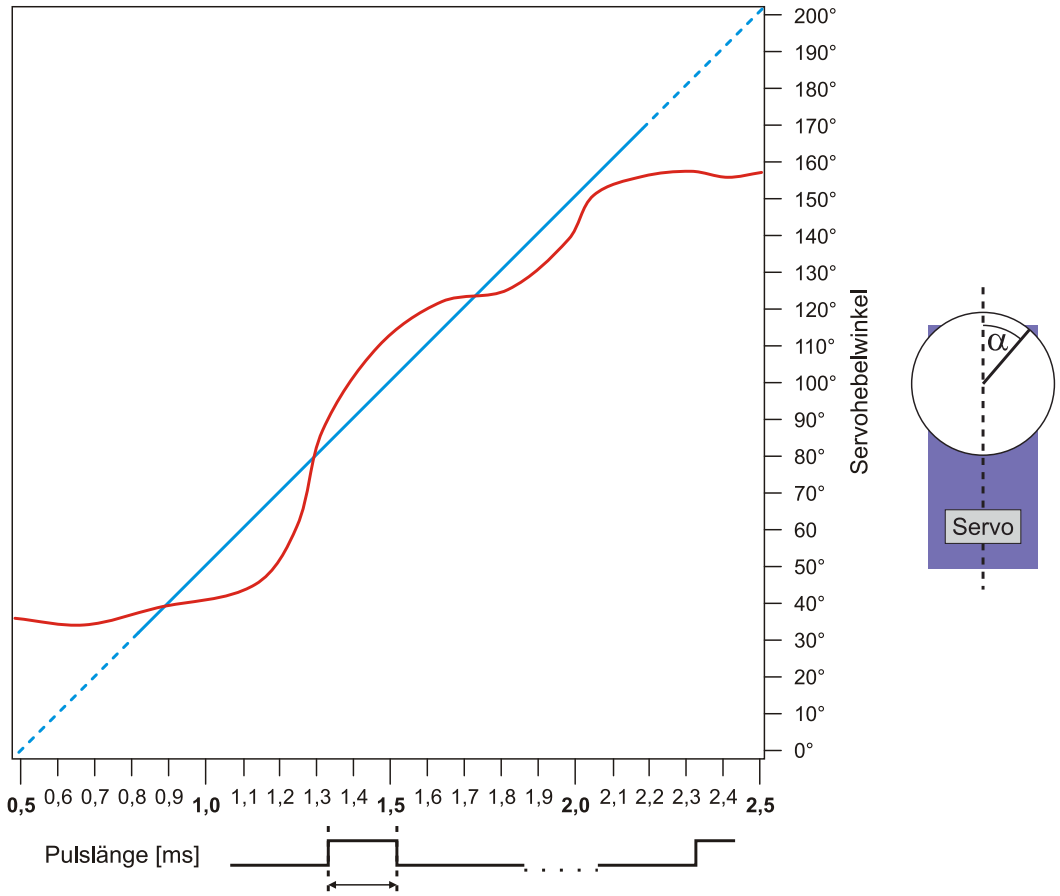


Abbildung 7.6: Gemessener Zusammenhang von Pulslänge (= Soll-Winkel) und resultierendem Servoarmeinstellwinkel bei einem idealen (blaue Gerade) und einem realen (rote gekrümmte Linie) Servo. Im Bild ist die Nichtlinearität des Zusammenhangs beim realen Servo zur Verdeutlichung etwas übertrieben dargestellt.

stelltem Winkel, die sogenannte Servokennlinie, gemessen worden. Die Abweichung von der idealen Kennlinie (blau) ist deutlich sichtbar.

Das Bein eines Laufroboters etwa wird in seinen Gelenken so abgewinkelt, dass der Fußpunkt eine bestimmte Raumkoordinate (x, y, z) erreicht. Die beschriebene Ungenauigkeit wirkt sich unmittelbar auf die tatsächlich vom Fußpunkt eingenommene Raumkoordinate aus. Der Roboter wird sich nur „ungefähr“ wie gewünscht bewegen, ähnlich wie sich ein Betrunkener nur ungefähr so wie eigentlich von ihm gewünscht bewegt.

Der HSSE-Robotics-Chip enthält eine Linearisierungseinheit, durch die sich die Ungenauigkeiten eines Servos in gewissem Rahmen ausgleichen lassen. Insbesondere die bei Billigprodukten ausgeprägten Serienschwankungen lassen sich durch diese Funktion in den Griff bekommen.

7.4 Servo-Anschlüsse

Die SandboxS 2 besitzt Anschlüsse für 24 Servos. Abbildung 7.7 zeigt den entsprechenden Ausschnitt des Schaltplans und ein Foto der Steckverbinder für die Servostecker.

Die Servostecker werden an alle Servoanschlüsse stets so angesteckt, dass sich die Seite auf welcher die vergoldeten Crimp-Kontakte im Kunststoffgehäuse des Servosteckers sichtbar sind auf der Oberseite (= Bestückungsseite der Platine) be ndet. Steckt man die Servostecker versehentlich falsch an (also um 180° verdreht), nehmen die Servos keinen Schaden, funktionieren selbstverständlich aber nicht.

Beim ersten Anstecken eines Servosteckers sind die Crimp-Kontakte meist noch etwas zu eng eingestellt. Daher muss der Stecker mit erhöhter Kraft und angemessener Vorsicht angesteckt werden.

Die Servos werden mit Pegeln nach dem LVCMOS-Standard (3,3 V) angesteuert; Versorgungsspannung der Servos ist jedoch die wesentlich höhere Akkuspannung. Nicht alle Servos auf dem Markt können mit der Akkunennspannung von 6 V betrieben werden. Fünf frisch geladene Akkuzellen, wie sie auf der SandboxS 2 verwendet werden, weisen in den ersten Betriebsminuten unter Umständen eine Spannung knapp oberhalb von 7 V auf. Ob ein Servo dies verkraftet oder nicht, sollte also vor dem Kauf bereits geprüft werden.

Die Lage der Stiftleisten am Platinenrand birgt gewisse Tücken in der Handhabung. Greifen Sie die Platine nie an diesen Stiftleisten! Es besteht ernsthafte **Verletzungsgefahr** durch die Stifte. Diese dringen sehr leicht durch die Haut. Auf unbenutzte Stiftleisen kann ein Streifen Styropor, Depron oder ähnliches nichtleitendes Material als Abdeckung aufgesteckt werden.

7.5 Servopulsgenerator

Aufgabe des Servopulsgenerators ist die Erzeugung genauer PWM-Pulssignale zur Ansteuerung der Servos. Der Servopulsgenerator ist in hohem Maße kon gurierbar, um das Erstellen der Software für den AVR-Mikrocontroller zu erleichtern. Die Erzeugung der Servopulssignale unterliegt darüber hinaus sehr engen Echtzeitanforderungen, die von einer Softwarelösung nur schwer mit der erwünschten Genauigkeit wahrgenommen werden können.

Abbildung 7.8 bietet einen Überblick des Servopulsgenerators und seiner Parameter.

7.5.1 Mode

Die Ansteuerung des Servopulsgenerators mit der gewünschten Pulslänge erfolgt im Modus PwmDirect aus dem Register PwmLen, das direkt vom Mikrocontroller aus beschrieben wird. Im wesentlich komfortableren Modus PhiSpeed wird die Pulslänge von der Linearisierungseinheit (Abschnitt 7.6) berechnet, die wiederum vom Servo-Sequencer (Abschnitt 7.7) automatisch mit einem Bewegungsablauf versorgt werden kann.

Der Modus PwmDirect ist hauptsächlich zur Kalibrierung der Servos und zu Debuggingzwecken gedacht.

7.5.2 Pulsunterdrückung

Mittels Bit 0 — Active in Register ModeIrqMask kann die Ausgabe eines Pulses unterdrückt werden. Ist dieses Bit mit dem Wert 0 beschrieben, so wird kein Puls ausgegeben. Dies ist auch der Initiali-

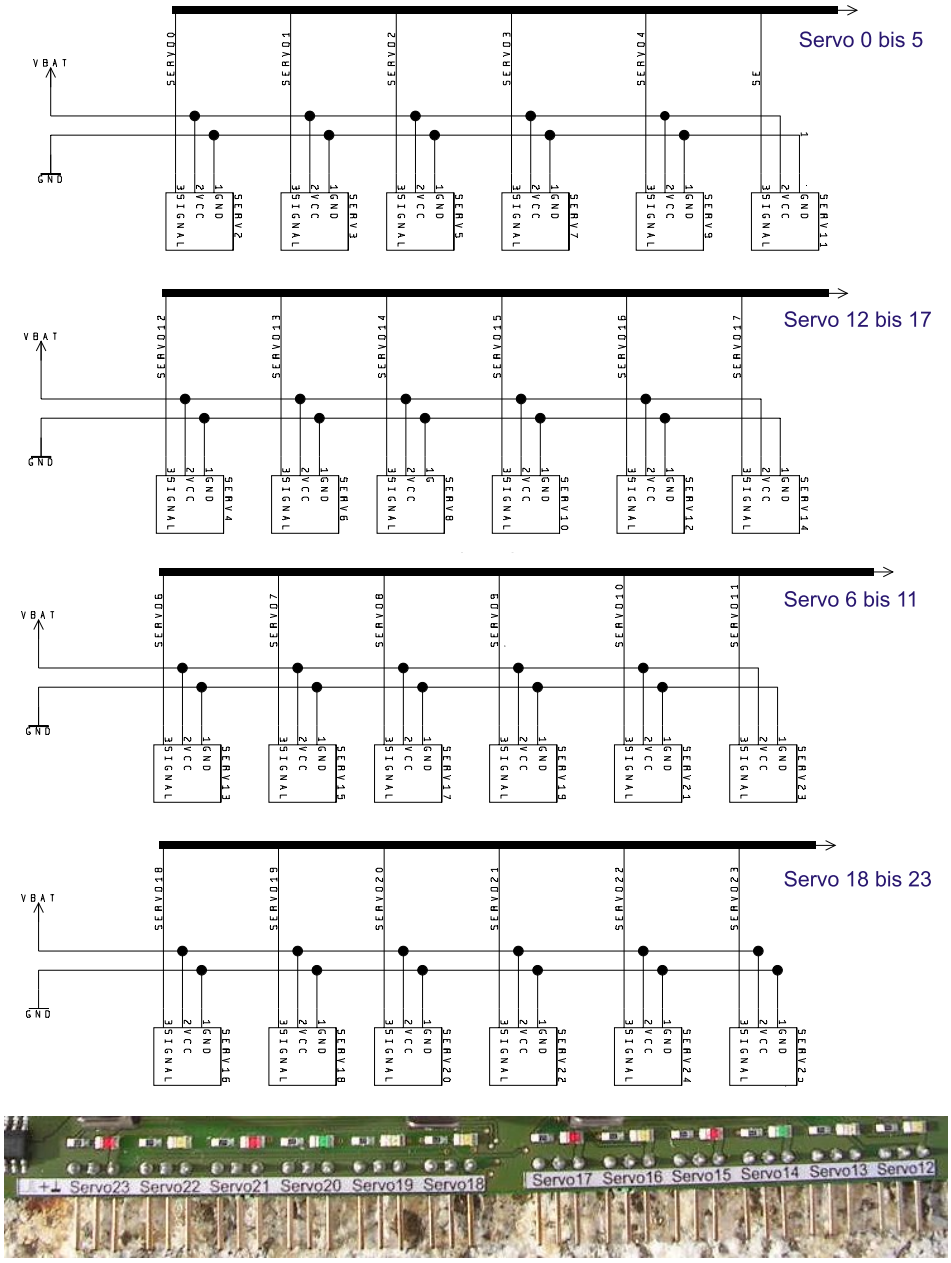


Abbildung 7.7: Servoanschlüsse am FPGA. Im Foto ist nur die Hälfte der Servoanschlüsse am unteren Platinenrand zu sehen. Die Anschlüsse für die Servos 0 bis 11 be nden sich am oberen Rand.

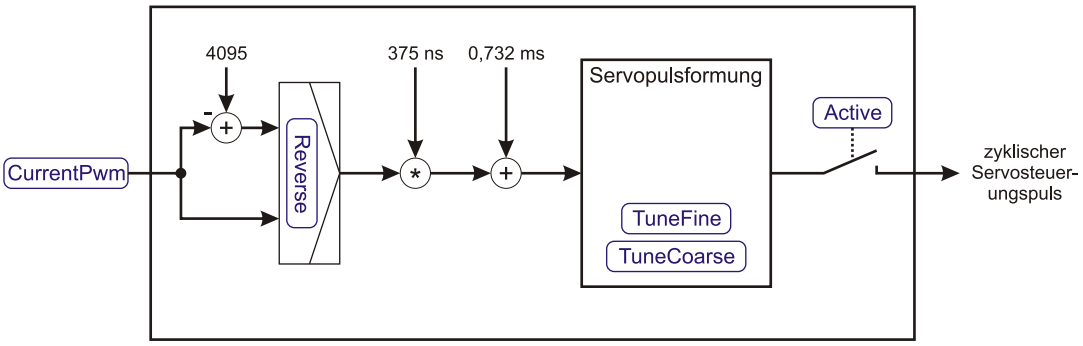


Abbildung 7.8: Der Servopulsgenerator und seine Parametrisierungsmöglichkeiten.

sierungswert nach dem Einschalten der SandboxS 2. Die meisten Servos werden in diesem Fall den Motor stromlos schalten. Ist das Bit mit dem Wert 1 beschrieben, wird zyklisch ein Impuls ausgegeben.

Nach dem Einschalten der SandboxS 2 sollten die Servos der Reihe nach aktiviert werden (Bit 0 — Active auf 1 setzen), wobei nach dem Aktivieren eines Servos lange genug gewartet (ca. 0,3–0,5 s) werden sollte, damit sich das Servo auf den gewünschten Winkel einstellen kann. Erst wenn dies der Fall ist, sollte das nächste Servo aktiviert werden. Diese Vorgehensweise schont die Robotermechanik, vermeidet einen schockartig einsetzenden enorm hohen Strom beim Einschalten und schaut außerdem auch erheblich eleganter aus.

7.5.3 Drehrichtungsumkehr

Servos können bei zunehmender PWM-Impulslänge entweder im Uhrzeigersinn oder in Gegenrichtung ausschlagen. Mit diesem Parameter kann ausgewählt werden, ob die vom Anwender eingegebene (oder von der Tabellenfunktion errechnete) PWM-Impulslänge verwendet wird oder ob stattdessen eine invertierte Impulslänge berechnet und eingesetzt wird. Die Berechnung der Impulslänge erfolgt, indem die ursprüngliche Impulslänge vom Maximum (4095) subtrahiert wird.

Die Drehrichtungsumkehr wirkt immer auf den die Länge des PWM-Pulses, welches das Servo steuert. Wird die Steuerung im Modus PhiSpeed (oder im Pseudomodus PhiDirect) über Winkelwerte vorgenommen, so kann mit der Drehrichtungsumkehr *nicht* das Vorzeichen des Stellwinkels gewählt werden! Vielmehr wird der richtige Wert für die Drehrichtungsumkehr im Rahmen des Kalibrierverfahrens (Abschnitt 7.6.1) ermittelt und bleibt dann für die gesamte Lebensdauer des betreffenden Servos xiert. Drehungen entgegen der mathematisch positiven Richtung, also im Uhrzeigersinn, werden durch negativer Winkelwerte repräsentiert.

7.5.3.1 Register ModeIrqMask

Bit	7	6	5	4	3	2	1	0
Adr+1	—	—	—	—	—	—	—	—
Adr	—	—	—	—	**ImSqe**	**Mode**	**Reverse**	**Active**

Tabelle 7.6: Register: ModeIrqMask. Adr=0x8440+2n_S, n_S ist die Servonummer (0 bis 23). Der Initialisierungswert nach dem Einschalten ist 0x0000. Les- und schreibbar.

Bit 3 — IrqMsk Ist dieses Bit gesetzt, so löst die Auswertung des zuvor vom Mikrocontroller geschriebenen Servosequenzauftrages (bestehend aus den Werten für PhiStart, DeltaPhi und NrOfDeltas) einen Interrupt auf der Leitung INT1 aus. Ein Interrupt wird nur ausgelöst, wenn sich das Servo im *Mode* PhiSpeed be ndet.

Bit 2 — Mode Dieser Eintrag stellt den Modus für das Servo ein. Folgende Modi sind möglich:

0 – PwmDirect Das Servo wird direkt durch die Eingabe der PWM-Impulslänge gesteuert.

1 – PhiSpeed Das Servo wird durch die Eingabe eines Anfangswinkels, eines Increment-/Decrementwinkels, um den sich der Winkel pro Zeitschritt verändert und die Anzahl der Zeitschritte gesteuert. Der Increment-/Decrementwinkel verändert den derzeitigen Winkel im Zeitabstand von ca. 20 ms (mittels TuneFine und TuneCoarse einstellbar). Soll ein Winkel eingestellt und gehalten werden, so wird der Incrementwinkel 0 angegeben. Mit dieser Einstellung steht quasi ein dritter Modus — PhiDirect — zur Verfügung.

Bit 1 — Reverse Die PWM-Impulslänge wird so umgerechnet, dass sich ein umgekehrter Drehsinn des Servos ergibt. Dieser Parameter wirkt in allen Modi auf die entweder vom Mikrocontroller direkt geschriebene Impulslänge oder auf diejenige Impulslänge, welche intern im Modus PhiSpeed ermittelt wurde.

Bit 0 — Active Ist dieses Bit gesetzt, so wird ein Servoimpuls ausgegeben, andernfalls wird kein Impuls ausgegeben. Je nach Bauart, werden die meisten Servos bei fehlendem Impuls das gleiche Verhalten zeigen, wie bei fehlender Energieversorgung.

7.5.4 Pulsdauer

Die Pulsdauer kann vom Mikrocontroller aus im Register PwmLen eingestellt werden. Der derzeit ablaufende Pulszyklus bleibt jedoch von einer Neueinstellung unberührt. Der neue Wert wird erst zu Beginn des nächsten Pulszyklus vom Pulsgenerator verwendet. Dieses Verfahren garantiert, dass keine Pulse von unerwünschter Länge beim Übergang erzeugt werden. Die momentan verwendete Pulsdauer kann bei Bedarf dem Register CurrentPwm entnommen werden.

7.5.4.1 Register PwmLen

Dieses Register steht nur im Modus PwmDirect (Bit 2 — Mode in Register ModeIrqMask trägt den Wert 0) zur Verfügung.

Das Register enthält eine vorzeichenlose 12-Bit-Zahl (0-4095). Diese Zahl wird mit PwmLenDirect bezeichnet. PwmLenDirect gibt die Länge des PWM-Impulses, welcher an das Servo geschickt wird unmittelbar an. Sie beträgt 0,476 ms+PwmLenDirect*500 ns. Da der Wertebereich von PwmLenDirect zwischen 0 und 4095 liegt, ist die Impulsdauer im Bereich von 0,476–2,523 ms einstellbar.

Bit	7	6	5	4	3	2	1	0
Adr+1	—	—	—	—	**unsigned(11 downto 8)**			
Adr	**unsigned(7 downto 0)**							

Tabelle 7.7: Register: PwmLen. Wertebereich: 0–4095. Adr=0x8480+2n_S, n_S ist die Servonummer (0 bis 23). Der Initialisierungswert nach dem Einschalten ist für alle Servos 0x0800 (2048), was einer Pulslänge von 1,5 ms (ungefähr Hebelmittelstellung) entspricht. Les- und schreibbar.

7.5.4.2 Register CurrentPwm

Aus diesem Register kann die Dauer des derzeit tatsächlich erzeugten PWM-Pulses entnommen werden. Dieses Register enthält eine vorzeichenlose 12-Bit-Zahl, welche das gleiche Format verwendet wie beim Register PwmLen.

Die Länge des PWM-Impulses wird unabhängig von der Einstellung des Parameters ServoReverse aus dem Register ModeIrqMask jedes Servos für die nicht drehrichtungsumgekehrte Servoeinstellung angegeben.

Bit	7	6	5	4	3	2	1	0
Adr+1	—	—	—	—	**unsigned(11 downto 8)**			
Adr	**unsigned(7 downto 0)**							

Tabelle 7.8: Register: CurrentPwm. Wertebereich: 0–4095. Adr=0x86C0+2n_S, n_S ist die Servonummer (0 bis 23). Nur lesbar.

7.5.5 Pulswiederholungszyklus

Die Dauer des Zyklus , in welchem der Puls wiederholt wird, kann mittels der Parameter TuneFine und TuneCoarse eingestellt werden. Standardeinstellung nach dem Einschalten der SandboxS 2 ist 20 ms. Die Einstellbarkeit der Zykluslänge steht im Zusammenhang mit der Abstimmungsfunktionen für die Geschwindigkeit, mit der Bewegungssequenzen vom Servo-Sequencer ausgeführt werden. Daher werden die Register für diese beiden Parameter im Abschnitt 7.7.1 besprochen.

7.6 Servolinearisierung und -kalibrierung

Für Robotikanwendungen ist die ungenaue und nichtlineare Zuordnung von Pulslänge zu Servostellwinkel handelsüblicher Servos sehr störend. Um sie auszugleichen, enthält der HSSE-Robotics-Chip eine Linearisierungseinheit, welche die nichtlineare Kennlinie eines Servos mit Hilfe von Stützstellen und zwischen diesen Stützstellen mittels linearer Interpolation ausgleicht.

Die Funktionsweise dieser Einheit ist in Abb. 7.9 dargestellt. In die Linearisierungseinheit wird ein Zahlenwert im Bereich von 0 bis 4095 eingegeben, der im Parametersatz des Servos als CurrentPhi, also momentaner Winkel bezeichnet wird. Allerdings handelt es sich hierbei nur um den gewünschten Winkel, nicht um den Winkel, der tatsächlich vom Servo eingenommen wird.

Die Linearisierungseinheit berechnet anschließend auf der Basis der Stützstellen Phi2PwmTable(0) bis Phi2PwmTable(16) mittels linearer Interpolation die passende Pulslänge, die im Register CurrentPwm abgelegt wird und aus diesem auch ausgelesen werden kann.

Der Wert von CurrentPhi kann vom Mikrocontroller nur ausgelesen werden. Einstellen lässt sich der gewünschte Winkel über den Parameter PhiStart, wenn der Modus PhiSpeed im Register ModeIrqMask ausgewählt wurde. Ist der Parameter NrOfDeltas auf den Wert 0 gesetzt, wird CurrentPhi direkt aus PhiStart übernommen.

Auf Wunsch kann der in Abschnitt 7.7 beschriebene Servo-Sequencer den Wert von CurrentPhi auch zum Erzielen einer gleichmäßig fortlaufenden Servobewegung automatisch nachführen.

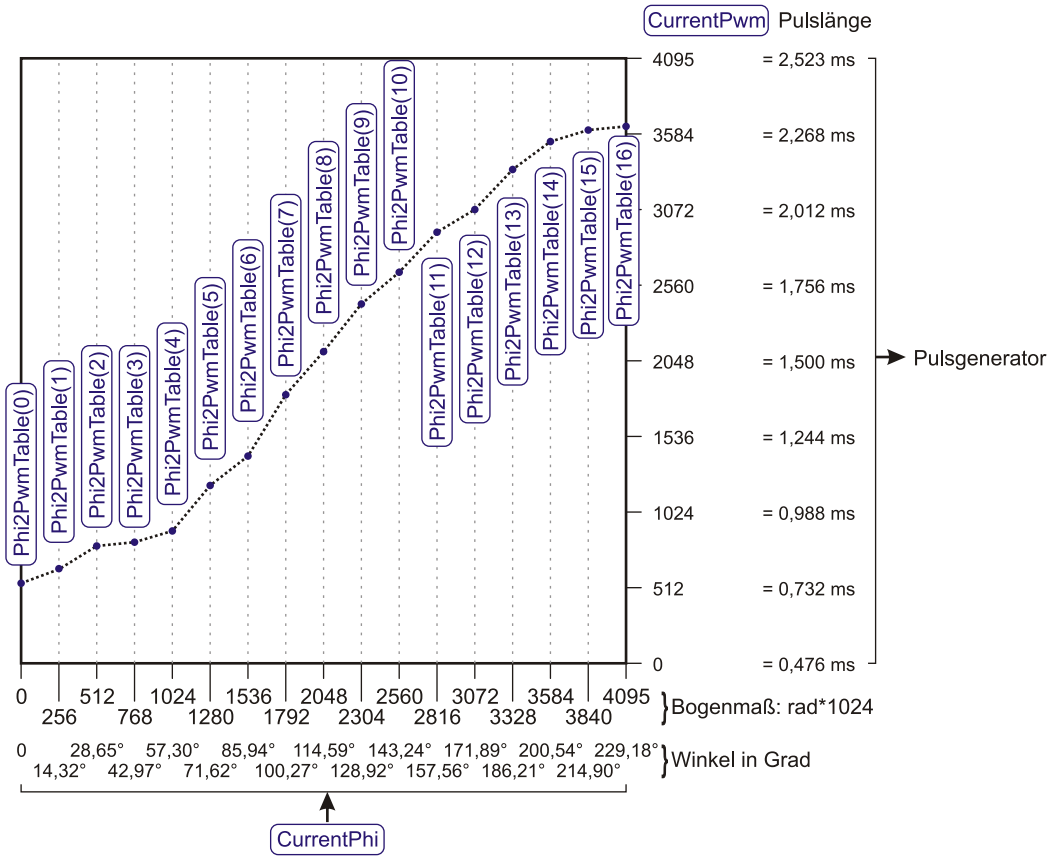


Abbildung 7.9: Linearisierungseinheit mit einstellbaren Stützstellen Phi2PwmTable(0) bis Phi2PwmTable(16). Die Stützstellen lassen sich entlang der strichlierten Linien verschieben, indem entsprechende Werte zwischen 0 und 4095 für jede der Stützstellen eingetragen werden. Die Steigung der Kurve muss an jedem Punkt größer oder gleich 0 sein. Folgewerte der Tabelle müssen also stets größer oder gleich ihrem Vorgängern sein. Werte, die zwischen den Stützstellen liegen, werden mittels linearer Interpolation errechnet. Insgesamt ergibt sich eine Umrechnungsfunktion, wie sie im Bild als Lininenzug dargestellt ist.

7.6.0.1 Registerfeld Phi2PwmTable(0) –Phi2PwmTable(16)

Die 17 Register Phi2PwmTable(0) bis Phi2PwmTable(16) enthalten die Werte der Funktionstabelle, welche zur Korrektur der Servokennlinie dient. Jeder Wert stellt eine vorzeichenlose 12 Bit breite Binärzahl dar. Höhere Tabelleneinträge müssen gegenüber niedrigeren Einträgen gleich oder größer sein. Wird gegen diese Regel verstoßen, kommt es zu internen Überläufen der Arithmetikeinheiten — unde niertes Verhalten ist die Folge. Um die gewünschte Drehrichtung des Servos einzustellen, kann der Parameter Reverse im Register ModeIrqMask (siehe Abschnitt 7.5.3.1 dienen. In Kombination mit der Linearisierungstabelle ist es möglich, praktisch aus jedem Modellbauservo ein Servo zu machen, dass sich zum einen bei steigenden Winkelwerten in mathematisch positiver Drehrichtung bewegt (also entgegen dem Uhrzeigersinn) und dass zum anderen eine genaue und direkte Einstellung des gewünschten Winkels erlaubt.

Bit	7	6	5	4	3	2	1	0
Adr+1	—	—	—	—	**unsigned(11 downto 8)**			
Adr	**unsigned(7 downto 0)**							

Tabelle 7.9: Register: Phi2PwmTable(0 - 16). Adr=0x8000+0x40*TableIdx+2n_S, n_S ist die Servonummer (0 bis 23), TableIdx kann Werte zwischen 0 und 16 annehmen. Der Initialisierungswert nach dem Einschalten ist unbestimmt. Les- und schreibbar.

Adr(AVR)	Ende	Wortzahl	Beschreibung	Power-On-Wert
0x8000+2n_S	0x803F	32	Phi2PwmTable(0) Servo n_S	51
0x8040+2n_S	0x807F	32	Phi2PwmTable(1) Servo n_S	362
0x8080+2n_S	0x80BF	32	Phi2PwmTable(2) Servo n_S	628
0x80C0+2n_S	0x80FF	32	Phi2PwmTable(3) Servo n_S	887
0x8100+2n_S	0x813F	32	Phi2PwmTable(4) Servo n_S	1158
0x8140+2n_S	0x817F	32	Phi2PwmTable(5) Servo n_S	1425
0x8180+2n_S	0x81BF	32	Phi2PwmTable(6) Servo n_S	1708
0x81C0+2n_S	0x81FF	32	Phi2PwmTable(7) Servo n_S	1996
0x8200+2n_S	0x823F	32	Phi2PwmTable(8) Servo n_S	2306
0x8240+2n_S	0x827F	32	Phi2PwmTable(9) Servo n_S	2598
0x8280+2n_S	0x82BF	32	Phi2PwmTable(10) Servo n_S	2875
0x82C0+2n_S	0x82FF	32	Phi2PwmTable(11) Servo n_S	3160
0x8300+2n_S	0x833F	32	Phi2PwmTable(12) Servo n_S	3425
0x8340+2n_S	0x837F	32	Phi2PwmTable(13) Servo n_S	3690
0x8380+2n_S	0x83BF	32	Phi2PwmTable(14) Servo n_S	3953
0x83C0+2n_S	0x83FF	32	Phi2PwmTable(15) Servo n_S	3953
0x8400+2n_S	0x843F	32	Phi2PwmTable(16) Servo n_S	3953

Tabelle 7.10: Addressraum für die Einträge der Linearisierungstabelle. n_S ist die Servonummer (0 bis 23).

Die Stützstellen sollten für jedes Servo einzeln mittels Messung bestimmt werden. Nur so wird eine hohe Genauigkeit erreicht. Man kommt jedoch schon durch Messung eines einzelnen Servoexemplars, dessen Werte der Einfachheit halber für alle Servos dieser Bauart eingesetzt werden, auf eine erhebliche Verbesserung der Genauigkeit.

7.6.1 Vorgehensweise für die Servokalibrierung

In Abschnitt 7.3 wurde bereits erläutert, warum käufliche (also reale) Servos nicht die Eigenschaften eines idealen Servos erreichen. Für die Robotik wäre es wünschenswert, das Servo direkt über einen Wunschwinkel im Bogenmaß, also beispielsweise $\pi/2$ steuern zu können. Der Winkel sollte dann vorzeigerichtig, also gegen den Uhrzeigersinn für wachsende Winkelwerte vom Servo angesteuert werden. Der Sinn der Linearisierungseinheit liegt nun genau darin, aus einem beliebigen Servo eines zu machen, dass dieses wünschenswerte Verhalten zeigt.

Der Winkelbereich 0°–205° entspricht in etwa dem maximal von gängigen Modellbauservos, wie dem Hitec HS311 erreichbaren Winkelbereich. Die Angabe des gewünschten Winkels sollte wie gesagt in der in der Mathematik üblicheren Art und Weise geschehen, nämlich im Bogenmaß. Im Bogenmaß entspricht der Winkel 180° dem Wert π und demnach der Winkel 205° dem Wert 3,58. Multipliziert man diesen (Maximal)Wert mit 1024, so erhält man 3663. Durch diese Skalierung lässt sich der Zahlenbereich einer zwölf Bit breiten Dualzahl von 0–4095 für das Bogenmaß ef zient nutzen. Die Multiplikation mit dem Faktor 1024 kann für eine Dualzahl mit sehr geringem Rechenaufwand durchgeführt werden, da $1024 = 2^{10}$ ist und daher durch eine Verschiebung der Zahl um 10 Stellen nach links ersetzt werden kann. Genauso kann die Division durch 1024 durch eine Verschiebung um 10 Stellen nach rechts ersetzt werden. Im C-Code kann trotzdem beispielsweise `Alpha * 1024` geschrieben werden, da der Compiler die Optimierungsmöglichkeit automatisch erkennt und nutzen wird.

Zwar bleibt ca. 10% des Zahlenbereichs (die Werte von 3664 bis 4095) für die Angabe des Winkels ungenutzt, aber die Auswirkungen auf die Stellgenauigkeit handelsüblicher Servos sind äußerst gering. Selbst bei Verwendung von Hochpreisservos weit jenseits der 100 €-Marke reicht die Auflösung des gewählten Bereiches leicht aus.

Die Kalibriereinrichtung

Die Kalibriereinrichtung kann leicht selbst angefertigt werden. Hierzu sind die beiden Teile A1 und A2 aus dem Hexapodbausatz Captain Ahab erforderlich. Die Kalibrierrosette druckt man selbst aus. Sie be ndet sich auf der vorletzten Seite diesese Dokuments. Auf ihrer Rückseite ndet sich eine Anleitung zum unverzerrten Drucken mit dem Acrobat Reader. Als letztes Bauteil für die Kalibriereinrichtung benötigt man noch eine einigermaßen stabile Kartonschachtel, die zumindest A4-Größe haben und 3,5 cm oder höher sein sollte.

Abbildung 7.10: Bau der Winkelkalibriereinrichtung: Links wurde ein Karton mit der Rosette mit Hilfe eines Allesklebers beklebt. Im rechten Foto ist der Ausschnitt für das Servo angebracht worden. Die bauchige Form des Kartons ist eine Folge der mangelhaften Optik der verwendeten Digitalkamera.

Abb. 7.10 zeigt die Kartonschachtel mit der aufgeklebten Winkelrosette. Im rechten Foto der Ab-

bildung ist zu sehen, wie der Ausschnitt für das Servo angebracht wird. Es sollte ein minimal über den inneren Rahmen hinausgehender Ausschnitt angebracht werden. Die eigentliche Positionierung des Servos wird nicht durch diesen Ausschnitt, sondern durch den aufgeklebten Rahmen A1 (Abb. 7.11 linkes Foto) vorgenommen.

Abbildung 7.11: Bau der Winkelkalibriereinrichtung: Im linken Foto ist das Bauteil A2 zur genauen Positionierung des Servos auf die Winkelrosette aufgeklebt worden. Die Bauteile für den Winkelzeiger sind im mittleren Foto zu sehen. Rechts der komplettierte Winkelzeiger.

Der Bau des Winkelzeigers ist im mittleren und rechten Foto in Abb. 7.11. Die benötigten Schrauben von 6,5 mm Länge liegen dem Bausatz des Roboters Captain Ahab bei. In Abb. 7.11 rechts ist der zusammengebaute Winkelzeiger zu sehen.

Der Kalibriervorgang

a) Zur Kalibrierung wird das Servo zunächst mit einem Aufkleber gekennzeichnet und dann in die Kalibriervorrichtung gesetzt. Mit einem passenden mehrfach gefalteten Papierstreifen kann das Servogehäuse im Rahmen A1 der Vorrichtung xiert werden. Allzu fest braucht es nicht eingeklemmt zu werden.

b) Um das Servo kalibrieren zu können wird ein Programm für den AVR auf der SandboxS 2 benötigt, mit dem der Wert im Register PwmLen verändert und ausgelesen werden kann. Hierzu ist beispielsweise das in Abschnitt 2.6 beschriebene Programm *CalibrateServo* geeignet.

   Das Programm sollte mit einer mittleren PwmLen, also etwas 2048 starten. Diese PwmLen kann leicht mit den Tasten eingestellt werden. Nun wird das Servo an den Anschluss Servo0 angeschlossen. Wird PwmLen vergrößert, so muss sich das Servo gegen den Uhrzeigersinn drehen. Ist dies nicht der Fall, muss die Drehrichtung durch gleichzeitiges Betätigen beider Tasten geändert werden. Dadurch wird das Bit Reverse im Register ModeIrqMask gesetzt, so dass der Drehsinn umgekehrt wird. Dies ist beispielsweise für Servos des Typs Hitec HS311, HS322HD und HS325HB notwendig.

c) Nun wird PwmLen so lange verringert, bis das Servo am mechanischen Anschlag angelangt ist oder aber der Wert von PwmLen 0 erreicht hat. Falls das Servo an den mechanischen Anschlag gebracht wurde, wird PwmLen wieder soweit erhöht, bis das Servo nicht mehr gegen den Anschlag arbeitet (= brummt).

d) Als nächstes wird der Winkelzeiger so aufgesteckt, dass er sich im Bereich des ersten Strahlenfeldes (also ca. 4 Uhr) be ndet. In Abb. 7.12 ist eine mögliche Situation nach Abschluss dieses Schrittes im linken Foto zu sehen.

Abbildung 7.12: Links: Das Servo ist im Uhrzeigersinn an den Anschlag gefahren worden. Dann wurde der Winkelzeiger so aufgesteckt, dass er sich innerhalb des ersten Strahlenfeldes be nden. Rechts: Der Zeiger wurde von der Ausgangsposition zum ersten erreichbaren Strahl gegen den Uhrzeigersinn (mathamatisch positiv) gedreht. Im Beispiel entschied der Zufall für die Strahlengruppe mit dem Kreissymbol.

e) Aus dieser Position wird PwmLen soweit erhöht, bis der erste erreichbare Strahl durch durch die Mitte der kleinen Öffnung im Zeiger verläuft. Das Symbol für diesen Strahl gilt für den gesamten Kalibriervorgang. Im Beispiel in Abbildung 7.12 rechts hat der Zufall sich für das Kreissymbol entschieden. Der so eingestellte Wert ist der erste Wert für die Linearisierungstabelle (Phi2PwmTable(0)).

f) Um die weiteren Werte für die Linearisierungstabelle herauszu nden, wird der Reihe nach auf alle Strahlen mit dem gewählten Symbol eingestellt. In Abb. 7.13 ist auf den dritten Strahl mit dem Kreissymbol eingestellt. Der so gefundene Wert von PwmLen ist der Tabelleneintrag für Phi2PwmTable(2).

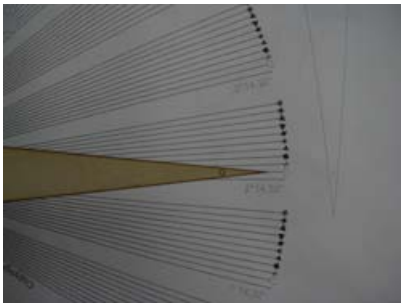


Abbildung 7.13: Die PwmLen (Abschnitt 7.5.3.1) wurde so eingestellt, dass der Strahl mit dem Kreissymbol im dritten Strahlenfeld erreicht wird. Der zugehörige Index in die Tabelle hat den Wert 2.

g) Die Winkelrosette überstreicht einen Bereich von annähernd 230°, welchen ein typisches Modellbauservo nicht gänzlich ausnutzen kann. Der mechanische Anschlag wird beim Hitec HS311 beispielsweise bei ca 205° erreicht. Daher bleiben Tabelleneinträge übrig, welche nicht mehr sinnvoll verwendbar sind. Diese werden mit dem gleichen Wert gefüllt, der für den letzten noch erreichbaren Strahl mit dem Symbol (im Beispiel wieder dem Kreis) gefunden wurde.

Ein Beispiel für eine typische Zahlenreihe, die auf diese Weise für ein Servo des Typs HS311 ermittelt wurde: 62, 382, 661, 944, 1225, 1508, 1787, 2082, 2384, 2680, 2924, 3181, 3424, 3662, 3898, 3898, 3898. Die Zahlen sind nach den Indizes 0 bis 16 geordnet. Der nutzbare Bereich geht offenbar nur bis zum Wert für den Index 14. Danach wird dieser Wert einfach nur wiederholt, dh.h. jeder Winkel in diesem Bereich wird zur Ausgabe des maximal gemessenen Wertes für PwmLen von 3898 führen.

7.7 Servo-Sequencer

Der Servo-Sequencer bewegt ein Servo im Modus PhiSpeed (Bit 2 — Mode in Register ModeIrqMask hat den Wert 1) automatisch mit gleichförmiger Geschwindigkeit. Die Bewegung erfolgt, ausgehend von einem einstellbaren Startwinkel, mit einer bestimmten Geschwindigkeit über eine bestimmte Dauer. Die drei Parameter Startwinkel (PhiStart), Geschwindigkeit (DeltaPhi) und Dauer (NrOfDeltas) können für jedes Servo einzeln angegeben werden.

Sequenzauftrag

Die Bewegungssequenz wird nicht unmittelbar von den drei Registern PhiStart, DeltaPhi und NrOfDeltas gesteuert. Vielmehr kann über diesen Parametersatz die auf die derzeit im Ablauf be ndliche Bewegungssequenz *folgende* Sequenz bestimmt werden. Über die Register wird quasi eine Bewegungssequenz in Auftrag gegeben. Dieser Auftrag wird aber erst nach vollständiger Abarbeitung der momentan ablaufenden Sequenz bearbeitet.

Der Status der momentan ablaufenden Sequenz lässt sich für jedes der Servos über die Register CurrentPhi (und CurrentPwm), CurrentDeltaPhi und DeltasToGo abfragen.

Eine neue Bewegungssequenz beginnt stets erst nach dem vollständigen Ablauf der vorigen Sequenz. Das Ende einer Sequenz ist am Wert 0 im Register DeltasToGo erkennbar.

Zu Beginn einer neuen Sequenz wird, wie in Abb. 7.14 zu sehen, der Wert aus PhiStart in das Regis-

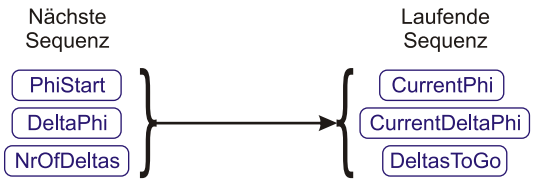


Abbildung 7.14: Registerinitialisierung zu Beginn einer Bewegungssequenz.

ter CurrentPhi geladen. Der Wert aus DeltaPhi wird in CurrentPhi geladen und die Anzahl der Servopulszyklen in DeltasToGo wird auf den Startwert aus NrOfDeltas gesetzt. Ist dies geschehen, so wird das Interrupt-Flag für dieses Servo im Interrupt-Flag-Register laut Abschnitt 7.2 gesetzt. Dieses Flag zeigt an, dass eine neue Sequenz vom Mikrocontroller angefordert werden darf. Das Interrupt-Flag wird in jedem Fall gesetzt und kann nur durch Auslesen des entsprechenden Interrupt-Flag-Registers wieder gelöscht werden. Ein Interrupt auf Leitung INT1 wird in Folge des gesetzten Interrupt-Flags jedoch nur dann ausgelöst, wenn das zugehörige Mask-Bit in Register ModeIrqMask (Abschnitt 7.5.3.1) gesetzt ist.

Interrupt für eine Servogruppe

Werden mehrere Servos mit Sequenzen im gleichen Rhythmus gesteuert, bedeutet dies typischerweise, dass für alle Servos Sequenzen mit identischen NrOfDeltas zum gleichen Startzeitpunkt vom Mikrocontroller aus angefordert werden. All diese Sequenzen werden praktisch gleichzeitig — nämlich unmittelbar nach Start der Sequenz — zum Setzen ihres jeweiligen Interrupt-Flags führen. Da die Servos von der Software im Mikrocontroller aus als Gruppe betrachtet werden, deren Mitglieder alle zum gleichen Zeitpunkt mit neuen Bewegungssequenzen beauftragt werden, reicht es aus, wenn nur das Interrupt-Flag eines einzigen Servo-Sequencers die Interruptleitung INT1 ansteuert. Dies kann leicht erreicht werden, indem nur für ein Servo aus der Gruppe das Interrupt-Mask-Bit gesetzt wird. Allerdings sollte hierbei beachtet werden, dass die Verarbeitung der Sequenzaufträge an die Servo

nicht parallel, sondern nacheinander in der Reihenfolge aufsteigender Servonummern erfolgt. Daher muss stets das Interrupt-Mask-Bit des Servo-Sequencers für das Servo mit der höchsten Nummer aktiviert werden. Sonst besteht die Gefahr, dass die Sequenzaufträge noch nicht für alle Servos abgeholt wurden. Aufträge, für die dies der Fall ist, würden bei Aufgabe von neuen Sequenzaufträgen also überschrieben.

Ablauf der Sequenz

Als Eingangswert der Linearisierungseinheit steuert der Wert in CurrentPhi (indirekt)mit die Ausgabe des Pulses an das Servo. Mit jedem neuen Servopulszyklus wird der Wert in CurrentPhi um den In-/Decrementwinkel CurrentDeltaPhi erhöht (oder bei negativem CurrentDeltaPhi verringert). Der Servopulszyklus wird durch den Pulsgenerator erzeugt. Die Dauer eines Zyklus kann über die Werte in den Registern TuneFine und TuneCoarse (siehe Abschnitt 7.7.1) verändert werden. Die dort getroffene Einstellung betrifft alle Servos in gleicher Weise.

Die Anzahl der Zyklen einer Bewegungssequenz wird dem Register NrOfDeltas entnommen. Hierdurch wird auch die Gesamtdauer der Sequenz bestimmt. Die Relation der verschiedenen Parameter

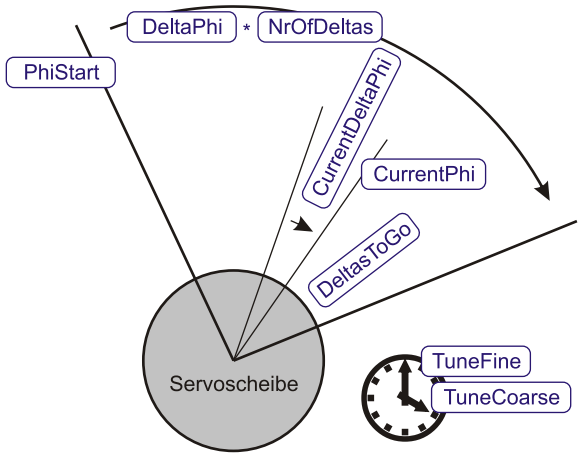


Abbildung 7.15: Eine Bewegungssequenz und ihre Parameter.

ist in der Graphik in Abbildung 7.15 verdeutlicht.

Rechtzeitig vor Ablauf einer Bewegungssequenz muss die nachfolgende Sequenz „bestellt“ werden. Dies kann frühestens geschehen, sobald das zu diesem Servo gehörende Interrupt-Flag gesetzt wurde. Wird vom Mikrocontroller keine neue Sequenz angefordert, so bleibt der zuletzt eingestellte Winkel des Servos erhalten. Eine vom Mikrocontroller geschriebene Bewegungssequenz wird also vom Sequencer quasi verbraucht. Der Winkel, mit dem die nachfolgende Sequenz beginnt, sollte möglichst genau mit dem Winkel übereinstimmen, der am Ende der momentan ablaufenden Sequenz erreicht wird, da sonst eine ruckartiger Übergang vom End- auf den Anfangswinkel mit maximaler Servostellgeschwindigkeit erfolgt.

Sonderfall: Winkelgeschwindigkeit 0 — Pseudomode „PhiDirect“

Wird das Register DeltaPhi mit dem Wert 0 beschrieben, entspricht dies der Bewegungsgeschwindigkeit 0. Folgerichtig bewegt sich das Servo dann unmittelbar auf den Startwinkel (PhiStart) und verbleibt dort. Das Register NrOfDeltas muss nicht zwingend mit 0 beschrieben werden. Daher können auf diese Weise die Servopulszyklen im Hintergrund gezählt werden. DeltasToGo wird also weiterhin

in jedem Servopulszyklus um eins verringert und auch das Interrupt-Flag wird am Ende der Sequenz gesetzt. Der Servo-Sequencer kann auf diese Weise auch als Timer verwendet werden.

7.7.0.1 Register PhiStart

Dieses Register steht nur im Modus PhiSpeed (Bit 2 — Mode in Register ModeIrqMask trägt den Wert 1) zur Verfügung.

Das Register enthält eine vorzeichenlose 12-Bit-Zahl (0–4095). Die Zahl im Register wird mit PhiStart bezeichnet. PhiStart gibt einen Servostellwinkel im Bereich von 0–4095 an. Die Bedeutung dieses Stellwinkels hängt von den Werten der Phi2PwmTable ab. Empfohlen wird die Skalierung nach
□ b Abschnitt

PhiStart dient nur als Ausgangswert für die Bewegung des Servos. Mittels der Werte von DeltaPhi und NrOfDeltas kann von diesem Wert ausgehend eine zeitlich lineare Winkelbewegung programmiert werden.

Bit	7	6	5	4	3	2	1	0
Adr+1	—	—	—	—	**unsigned(11 downto 8)**			
Adr	**unsigned(7 downto 0)**							

Tabelle 7.11: Register: PhiStart. Wertebereich: 0–4095. Adr=0x8480+2n_S, n_S ist die Servonummer (0 bis 23). Der Initialisierungswert ist für alle Servos Mittelstellung !!!. Les- und schreibbar.

□ c

Mittelstellung berechnen

7.7.0.2 Register CurrentPhi

Dieses Register enthält eine vorzeichenlose Festkommazahl im Format 12.4, also mit 12 Vorkommastellen und 4 Nachkommastellen. Werden die Nachkommastellen nicht gebraucht, so kann der Wert aus dem Register einfach um vier Stellen nach rechts geschoben werden. Der Wert, der aus dem Register ausgelesen werden kann, gibt den derzeit vom Servo-Sequencer vorgesehenen Servostellwinkel an.

Wird auf das Register vom Mikrocontroller geschrieben, kommt es zu unde niertem Verhalten. Das Register sollte daher ausschließlich gelesen werden.

Bit	7	6	5	4	3	2	1	0
Adr+1	**unsigned(11 downto 4)**							
Adr	**unsigned(3 downto -4)**							

Tabelle 7.12: Register: Current Phi. Dieses Register hat nur im *Mode* PhiSpeed eine Bedeutung. Adr=0x8680+2n_S, n_S ist die Servonummer (0 bis 23). Nur lesbar.

7.7.0.3 Register DeltaPhi

Dieses Register enthält eine vorzeichenbehaftete Festkommazahl im Format 10.4, also mit 10 binären Vorkommastellen (inkl. Vorzeichen) und 4 binären Nachkommastellen. Dieses Format erlaubt eine hohe Genauigkeit bei der Steuerung der Servobewegung. Der AVR-Mikrocontroller kennt dieses Zahlenformat nicht und kann lediglich vorzeichenbehaftete Ganzzahlen von -8192 bis +8191 im Zweierkomplementformat in das Register schreiben. Der HSSE-Robotics-Chip dividiert diese Zahlen durch 16, bevor sie als Winkelinkrement verwendet werden. Damit wird der kleinste Inkrementwert zu 0,0625

und die Bewegung des Servos über den gewünschten Winkelbereich wird mit hoher Präzision durchgeführt. Negative Zahlen stellen statt eines Winkelinkrements ein Winkeldekrement dar.

Bit	7	6	5	4	3	2	1	0
Adr+1	—	—	**signed(9 downto 4)**					
Adr	**signed(3 downto -4)**							

Tabelle 7.13: Register: DeltaPhi. Dieses Register hat nur im *Mode* PhiSpeed eine Bedeutung. Adr=0x84C0+2n_S, n_S ist die Servonummer (0 bis 23). Initialisierungswert nach dem Einschalten ist 0x0000. Les- und schreibbar.

7.7.0.4 Register CurrentDeltaPhi

Dieses Register enthält eine vorzeichenbehaftete Festkommazahl im Zweierkomplementformat 10.4, also mit 10 Vorkommastellen (inkl. Vorzeichen) und 4 Nachkommastellen. Damit ist das Format der enthaltenen Zahl identisch zum Zahlenformat des Registers DeltaPhi. Das Register CurrentDeltaPhi wird zu Beginn einer Bewegungssequenz automatisch aus dem Inhalt des Registers DeltaPhi aktualisiert.

Wird auf das Register vom Mikrocontroller geschrieben, kommt es zu unde niertem Verhalten. Das Register sollte daher ausschließlich gelesen werden.

Bit	7	6	5	4	3	2	1	0
Adr+1	—	—	**signed(9 downto 4)**					
Adr	**signed(3 downto -4)**							

Tabelle 7.14: Register: Current DeltaPhi. Dieses Register hat nur im *Mode* PhiSpeed eine Bedeutung. Adr=0x8640+2n_S, n_S ist die Servonummer (0 bis 23). Nur lesbar.

7.7.0.5 Register NrOfDeltas

Dieses Register enthält eine vorzeichenlose 12-Bit-Zahl. Mit dieser Zahl wird die Anzahl der Winkelschritte angegeben, aus der die nächste Bewegungssequenz bestehen wird.

Bit	7	6	5	4	3	2	1	0
Adr+1	—	—	—	—	**unsigned(11 downto 8)**			
Adr	**unsigned(7 downto 0)**							

Tabelle 7.15: Register: NrOfDeltas. Dieses Register hat nur im *Mode* PhiSpeed eine Bedeutung. Adr=0x8500+2n_S, n_S ist die Servonummer (0 bis 23). Initialisierungswert nach dem Einschalten ist 0x0000. Les- und schreibbar.

7.7.0.6 Register DeltasToGo

Dieses Register enthält eine vorzeichenlose 12-Bit-Zahl. Diese Zahl gibt an, wie viele Winkelschritte in der momentan ablaufenden Bewegungssequenz noch übrig sind und wird zu diesem Zweck bei jedem Winkelschritt automatisch vom Sequencer dekrementiert.

Dieses Register kann dazu verwendet werden, zeitgerecht eine neue Bewegungssequenz zu initiieren. Hierzu kann beispielsweise gewartet werden, bis das Register mit dem Wert Null das Ende der

Sequenz anzeigt. Es ist jedoch möglich, schon unmittelbar nach dem Ablauf des ersten Winkelschrittes (DeltasToGo enthält einen Schritt weniger als NrOfDeltas) eine neue Sequenz zu starten, da die Werte der neuen Sequenz ohnehin erst nach dem vollständigen Ablauf der momentan in Arbeit be ndlichen Sequenz vom Sequencer geladen werden.

Wird auf das Register vom Mikrocontroller geschrieben, kommt es zu unde niertem Verhalten. Das Register sollte daher ausschließlich gelesen werden.

Bit	7	6	5	4	3	2	1	0
Adr+1	—	—	—	—	**unsigned(11 downto 8)**			
Adr	**unsigned(7 downto 0)**							

Tabelle 7.16: Register: DeltasToGo. Dieses Register hat nur im *Mode* PhiSpeed eine Bedeutung. Adr=0x8600+2n_S, n_S ist die Servonummer (0 bis 23). Nur lesbar.

7.7.1 Geschwindigkeitsabstimmung

Der Ablauf der Servobewegungssequenzen verwendet die Pulszyklen der Servos als Zeiteinheit für einen Winkelschritt. Die Dauer dieser Pulszyklen ist für alle Servos die gleiche. Sie kann im Bereich von 18,37 bis 29,2 ms eingestellt werden. Die Grundeinstellung nach dem Einschalten liegt bei 20 ms.

Durch Justieren dieser Zyklusdauer können die Bewegungssequenzen auch im Nachhinein noch leicht in ihrer Ablaufgeschwindigkeit verändert werden, beispielsweise um sie genau an das Tempo eines Musikstückes anzupassen.

7.7.1.1 Register TuneFine

Mit diesem Register ist die Dauer des Servopulszyklus in Schritten von 0,50 μs stimmbar. Die Grundeinstellung des Registers nach dem Einschalten erzeugt einen 20 ms langen Zyklus. Der zugehörige Registerwert ist 1619 (0x0653). Der Wert 0x000 entspricht einer Verringerung um 0,8095 ms. Der Wert 0x0FFF entspricht einer Erhöhung um 1,239 ms.

Da die Zykluslänge von der Genauigkeit des Taktgebers auf der SandboxS 2 abhängig ist, wird eine Zykluszeit von exakt 20 ms im allgemeinen nicht mit dem Standardwert, sondern bei einem leicht abweichenden Wert für TuneFine erreicht.

Bit	7	6	5	4	3	2	1	0
0x85F7	—	—	—	—	**unsigned(11 downto 8)**			
0x85F6	**unsigned(7 downto 0)**							

Tabelle 7.17: Register: TuneFine. Wertebereich: 0–4095. Initialwert nach dem Einschalten: 0x0653 (1619). Les- und schreibbar.

7.7.1.2 Register TuneCoarse

Mit diesem Register ist die Dauer des Servopulszyklus in Schritten von 2,5 μs stimmbar. Die Grundeinstellung des Registers nach dem Einschalten erzeugt einen 20 ms langen Zyklus. Der zugehörige Registerwert ist 1619 (0x0653). In der Minimaleinstellung dieses Parameters (0x0000) wird die Zykluszeit um 4,05 ms verkürzt. Mit der Maximaleinstellung für den Paramter TuneCoarse kann die Zykluszeit um 6,19 ms gedehnt werden. Möglicherweise bereitet diese lange Zykluszeit bei einigen

Servotypen Probleme bei der Impulsauswertung. In der Praxis sind solche Probleme jedoch mit keinem der getesteten Servotypen aufgetreten.

Bit	7	6	5	4	3	2	1	0
0x85F9	—	—	—	—	**unsigned(11 downto 8)**			
0x85F8	**unsigned(7 downto 0)**							

Tabelle 7.18: Register: TuneCoarse. Wertebereich: 0–4095. Initialwert nach dem Einschalten: 0x026B (619). Les- und schreibbar.

7.8 Servogrundstellungsmodus

Durch Druck der Taste KeyFpga1 kann der Servogrundstellungsmodus gewählt werden. Erneuter Druck auf die Taste verlässt diesen Modus wieder. Im Servogrundstellungsmodus leuchtet die LED MsgFpga1 auf.

Der Servogrundstellungsmodus dient dazu, die Servos beispielsweise für den korrekten Einbau in eine Robotermechanik in ihre Mittelstellung zu bringen. Die Servomittelstellung wird eingestellt, indem alle Servos mit einer Impulslänge von 1,5 ms angesteuert werden. Diese Impulslänge wir jedoch nur erreicht, wenn die Werte der beiden Register TuneFine und TuneCoarse die gleichen sind, wie nach dem Einschalten der SandboxS 2. Auf Grund der Toleranzen der einzelnen Servos wird die mechanische Mittelstellung des Servos meist auch bei einer Impulslänge von 1,5 ms nicht genau erreicht.

Je nach Einbauort des Servos, ist es häu g auch notwendig, das Servo in eine der Extremstellungen zu bringen. In den extremen Winkeleinstellungen verfügen Modellbauservos üblicherweise über mechanische Wegbegrenzungen, so dass die mechanischen Endstellungen durch sanftes und langsames Verdrehen der Servoscheibe von Hand erreicht werden können. Der extrem hohe Pulsbreitenbereich, der sich mit dem Servopulgenerator des Robotics-Chips erreichen lässt, führt dazu, dass das Servo auch auf elektrischem Wege von Anschlag bis Anschlag bewegt werden kann. Allerdings geschieht dies typischerweise sehr unsanft und sollte besser vermieden werden. Entsprechende Mittel stellt der Robotics-Chip mit den Parametern PhiMin und PhiMax, sowie PhiOffset (siehe Abschnitt 7.6) zur Verfügung.

Da alle angeschlossenen Servos nach Drücken der Taste KeyFpga1 praktisch ohne Verzögerung in die durch den Servogrundstellungsmodus ausgewählte Stellung fahren, ist bei der Auswahl dieses Modus Vorsicht geboten! Bereits eingebaute Servos treffen womöglich auf einen mechanischen Anschlag im Roboter. Spezielle Roboterservos sind teilweise so drehmomentstark, dass die Robotermechanik hierbei beschädigt werden kann. Auch Verletzungen durch die Servobewegung sind bei sehr kräftigen Servos nicht ausgeschlossen!

Beim Übergang in den Servogrundstellungsmodus wird das zugehörige Interrupt-Flag IFNeut im Register IrqFlagsOthers (siehe Abschnitt 7.2.1.4 gesetzt. Ein Interrupt wird auf der Leitung INT0 nur dann ausgelöst, wenn das Interrupt-Mask-Bit ImNeut in Register IrqMaskRcRxCtrl (Abschnitt 7.13.1.1 gesetzt ist (Wert 1).

7.9 Servopaniktaste

Druck auf die Taste KeyFpga0 deaktiviert die Impulserzeugung aller angeschlossenen Servos. Der Effekt ist der gleiche, wie wenn das Bit Active des Registers ModeIrqMask aller Servos deaktiviert

würde. Die Deaktivierung kann nur durch Druck auf die Taste Reset oder Aus- und Einschalten wieder rückgängig gemacht werden.

Beim Drücken der Servopaniktaste wird das zugehörige Interrupt-Flag IFPnk im Register IrqFlags-Others (siehe Abschnitt 7.2.1.4 gesetzt. Ein Interrupt wird auf der Leitung INT0 nur dann ausgelöst, wenn das Interrupt-Mask-Bit ImPnk in Register IrqMaskRcRxCtrl (Abschnitt 7.13.1.1 gesetzt ist (Wert 1).

Diese Taste hat noch eine Zweitbelegung, die zum Aufruf des Bootloaders im AVR-Mikrocontroller dient. Eine Beschreibung ndet sich in Abschnitt 4.7.1. In dieser Zweitbedeutung wird das Interrupt-Flag IFPnk nicht gesetzt.

7.10 LED mit Helligkeitssteuerung

Am FPGA sind 24 farbig sortierte LED zur Anzeige beliebiger Statusmeldungen (oder auch als Schmuck) angeschlossen (siehe Abb. 7.16). Deren Helligkeit kann in 256 Stufen eingestellt werden.

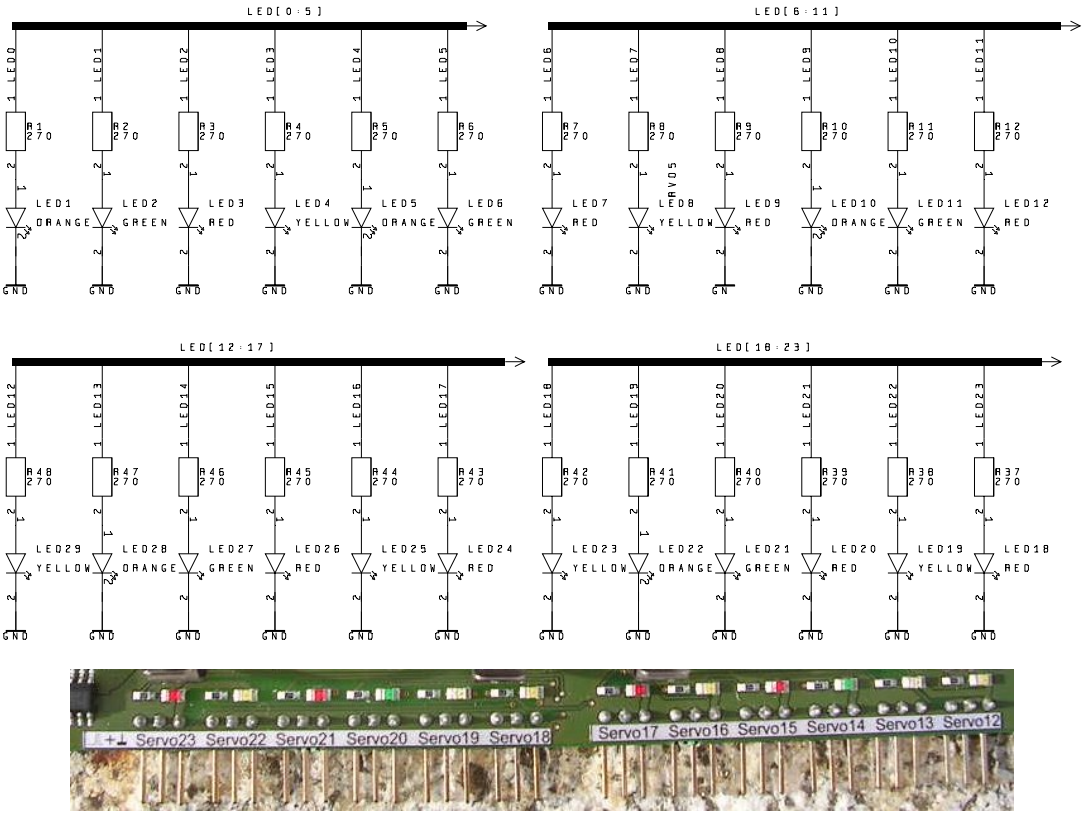


Abbildung 7.16: LEDs mit Helligkeitssteuerung am FPGA. Es ist nur die am unteren Platinenrand be ndliche Hälfte der LEDs abgebildet. Die LEDs 0 bis 11 be nden sich auf der gegenüberliegenden Seite der Platine am oberen Rand.

Auf Stufe 0 ist die LED vollkommen dunkel. Die Helligkeit der LED nimmt auf Grund der physikalischen Funktionsweise von LEDs nicht linear mit dem eingestellten Wert zu. Die Leuchtstärke der LEDs in unterschiedlichen Farben unterscheidet sich sehr stark voneinander.

Der Aufbau des Parametersatzes ist für alle LEDs identisch. Nach dem Einschalten ist die Helligkeit der LEDs so eingestellt, dass sich ein typischer Helligkitsverlauf ergibt. Um die LEDs auszuschalten oder andere Helligkeitswerte einzustellen, muss dieser Parameter möglichst frühzeitig im Programmablauf eingestellt werden.

7.10.0.3 Register LedBrightness

Bit	7	6	5	4	3	2	1	0
Adr+1	**CRM: ASCII-Zeichen(7 downto 0)**							
Adr	**LedBrightness: unsigned(7 downto 0)**							

Tabelle 7.19: Register: LedBrightness. Adr=0x8540+2n_L, n_L ist die LED-Nummer (0 bis 23). Les- und schreibbar.

Bits 15 bis 8 — CRM Nach dem Einschalten ist hier ein ASCII-Zeichen lesbar. Die Zeichen aller 24 LEDs hintereinander gestellt ergeben die Copyright-Message (CRM) für den HSSE-Robotics-Chip.

Bits 7 bis 0 — LedBrightness Vorzeichenlose 8-Bit-Zahl. Die Einstellung 0 bringt die LED zum Verlöschen. Die Einstellungen 1 bis 255 führen zu steigender Helligkeit der LED.

7.11 Status-LED

Neben den 24 in der Helligkeit steuerbaren LEDs sind zwei LEDs zur Anzeige von Statusinformationen am FPGA angeschlossen (Abb. 7.17). Der HSSE-Robotics-Chip gibt auf diesen LED folgende

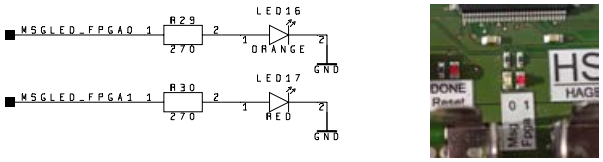


Abbildung 7.17: Status-LED am FPGA.

Informationen aus:

LED16, MsgFpga0 Leuchtet nach Druck auf die Servopaniktaste (KeyFpga0). Damit wird angezeigt, dass derzeit kein Servoimpuls ausgegeben wird (siehe Abschnitt 7.9).

LED17, MsgFpga1 Leuchtet im Servogrundstellungsmodus (siehe Abschnitt 7.8).

7.12 Taster

Am FPGA sind zwei Tasten laut Abb. 7.18 angeschlossen. Der HSSE-Robotics-Chip belegt diese mit folgenden Funktionen:

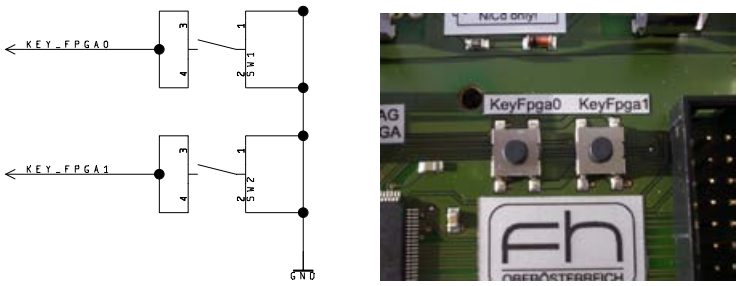


Abbildung 7.18: Taster am FPGA.

KeyFpga0 In Zusammenhang mit der Reset-Taste kann die Hardware-Boot-Funktion des AVRs aktiviert werden. Mehr hierzu im Abschnitt 6.2.1.

Außerhalb des Reset deaktiviert ein Druck auf die Taste sämtliche Servos (siehe Abschnitt 7.9).

KeyFpga1 Diese Taste ermöglicht den Wechsel in den Servogrundstellungsmodus und zurück. Eine genaue Beschreibung ndet sich im Abschnitt 7.8.

7.13 RC-Empfängerdecoder

7.13.1 RC-Empfängeranschluss

Der RC-Empfängeranschluss (RC steht für *Remote Control*, also Fernsteuerung) dient zum Einleiten des PPM-Signals, wie es nach der Demodulationsstufe eines handelsüblichen RC-Empfängers zur Verfügung steht. Auf diese Weise können bis zu 12 Kanäle empfangen werden, wobei die Kanalzahl des Empfängers irrelevant ist. Empfangen werden auf diese Weise stets alle Kanäle des Senders. Dies gilt auch, wenn der Empfänger eine andere Kanalzahl als der Sender hat. AM-Empfänger sind jedoch nicht mit FM-Sendern mischbar, da die Demodulationsstufe des Empfängers nicht für die Demodulation von FM-Signalen (FSK) geeignet ist. Angeschlossen wird der Empfänger an die Buchse in Bild 7.19.

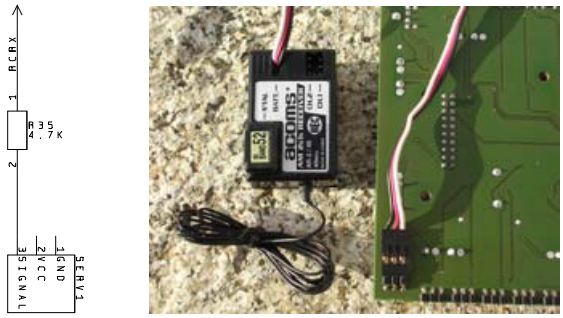


Abbildung 7.19: Anschluss für das Signal aus der Demodulationsstufe eines RC-Empfängers.

Bei den meisten Empfängern steht dieses Signal nicht auf einem von außen zugänglichen Anschluss zur Verfügung, sondern muss mit dem Oszilloskop im Empfänger aufgespürt werden. Als Beispiel ist in Abb. 7.20 das Oszilloskopbild für den Empfang eines 3-Kanal-Senders zu sehen. Meist ist einer der

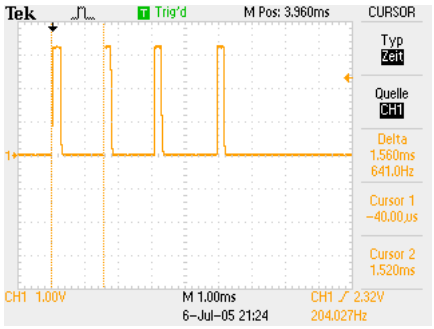


Abbildung 7.20: Oszilloskopbild mit dem PPM-Signal am Ausgang der Demodulationsstufe eines Fernsteuerempfängers bei Empfang eines 3-Kanal-Senders.

drei für die Stromversorgung verwendeten Pins frei und kann mit diesem Signal belegt werden. Alternativ wird an Stelle des Stromversorgungssteckers ein Servokabel direkt innerhalb des Empfängers angelötet und aus dem Empfänger herausgeführt, was etwas Gewicht spart und kostengünstiger ist.

Eine Umbauanleitung für einen handelsüblichen Empfänger enthält Anhang C.

Der Impulsausgang des Empfängers wird über einen Serienwiderstand zur Pegelanpassung auf einen FPGA-Eingang gegeben. Die eventuell notwendige Pegelanpassung wird bei zu hohem Eingangspegel von der Schutzdiode im FPGA vorgenommen.

7.13.1.1 Register IrqMaskRcRxCtrl

Dieses Register gibt Einstellungen für die Auswertung des PPM-Impulszugs eines RC-Empfängers an.

Bit	7	6	5	4	3	2	1	0
0x85F5	—	—	—	—	**ImCord**	**ImNeut**	**ImPnk**	**ImPs2**
0x85F4	—	—	**ImRc**	**InvPpm**	**NrOfChannels: unsigned(3 downto 0)**			

Tabelle 7.20: Register: IrqMaskRcRxCtrl. Alle Interrupt-Mask-Bits und InvPpm sind nach dem Einschalten deaktiviert (0), die Anzahl der Kanäle ist 2. Les- und schreibbar.

Bit 11 — ImCord Interrupt-Mask-Bit das die Auslösung eines Interrupts (INT0) nach der Fertigstellung eines Berchnungsauftrages des CORDIC-Prozessors (siehe Abschnitt 7.22) erlaubt (Wert 1) oder untersagt (Wert 0).

Bit 10 — ImNeut Interrupt-Mask-Bit das die Auslösung eines Interrupts (INT0) beim Aktivieren der Servogrundstellung (siehe Abschnitt 7.8) erlaubt (Wert 1) oder untersagt (Wert 0).

Bit 9 — ImPnk Interrupt-Mask-Bit das die Auslösung eines Interrupts (INT0) beim Aktivieren der Paniktaste (siehe Abschnitt 7.9) erlaubt (Wert 1) oder untersagt (Wert 0).

Bit 8 — ImPs2 Interrupt-Mask-Bit das die Auslösung eines Interrupts (INT0) beim Eintreffen neuer Daten von der PS/2-Schnittstelle (siehe Abschnitt 7.14) erlaubt (Wert 1) oder untersagt (Wert 0).

Bit 5 — ImRc Interrupt-Mask-Bit das die Auslösung eines Interrupts (INT0) beim Eintreffen neuer Daten vom RC-Empfänger erlaubt (Wert 1) oder untersagt (Wert 0).

Bit 4 — InvPpm Die meisten Fernsteuersender und -empfänger arbeiten mit sogenannten nichtinvertierten PPM-Signalen. Hierzu ist der Wert 0 einzustellen, was auch der Einstellung nach dem Einschalten entspricht. Einige Fabrikate verwenden jedoch ein invertiertes Puls-Pausen-Signal. In diesem Fall muss InvPpm auf den Wert 1 eingestellt werden.

Bits 3 bis 0 — NrOfChannels Diese vier Bit breite vorzeichenlose Binärzahl gibt die Anzahl der Kanäle an, welche der verwendete Sender hat. Die Einstellung nach dem Einschalten ist 0x2 (2).

7.13.1.2 Registerblock RcRxChannel(1) bis RcRxChannel(12)

Dieser Registerblock gibt die vom Fernsteuerempfänger empfangenen Pulslängen für die Senderkanäle 1 bis 12 jeweils mit einer 12 Bit breiten vorzeichenlosen Zahl an. Die Kanalzahl des Senders muss zuvor im Register RcRxCtrl korrekt eingestellt worden sein. Das Format, in dem die Dauer der empfangenen Pulse angegeben wird, ist das gleiche, das auch beim Register PwmLen (Abschnitt 7.5.4.1) eingesetzt wird.

Das Eintreffen eines vollständigen Satzes neuer Werte (also Werte für alle Kanäle) wird mittels des Interrupt-Flags IFRc im Interrupt-Flag-Register IrqFlagsOthers (Abschnitt 7.2.1.4 angezeigt. Ob die neuen Werte sich von den vorigen unterscheiden ist dabei nicht relevant. Dieses Interupt-Flag löst einen Interrupt auf Leitung INT0 aus, wenn das Interrupt-Mask-Bit ImRc im Register IrqMaskRcRxCtrl (Abschnitt 7.13.1.1) gesetzt ist (Wert 1).

Bit	7	6	5	4	3	2	1	0
Adr+1	—	—	—	—	**unsigned(11 downto 8)**			
Adr	**unsigned(7 downto 0)**							

Tabelle 7.21: Register: RcRxChannel(1) bis RcRxChannel(12). Wertebereich: 0–4095. Adr=0x8D8+2n_C, n_C ist die Kanalnummer (1 bis 12). Initialwert nach dem Einschalten: 0x0000. Nur lesbar.

7.13.1.3 Register RcRxStatus

Dieses Register gibt Informationen über die derzeitige Empfangslage des Fernsteuerempfängers, so dass beurteilt werden kann, ob der Empfang gestört oder aber einwandfrei ist.

Bit	7	6	5	4	3	2	1	0
0x87F1	—	—	—	—	—	—	—	—
0x87F0	—	—	—	—	—	—	—	**RxOk**

Tabelle 7.22: Register: RcRxStatus. Der Initialisierungswert nach dem Einschalten ist 0x0000. Nur lesbar.

Bit 0 — RxOk Dieses Bit trägt den Wert 0, wenn der Empfang des Fernsteuerempfängers gestört ist. Kann der RC-Empfängerdecoder das empfangene Signal decodieren, so ist dieses Bit mit dem Wert 1 versehen. In diesem Fall ist der Empfang mit hoher Wahrscheinlichkeit einwandfrei.

7.14 PS/2-Mausinterface

Am PS/2-Anschluss können handelsübliche Mäuse angeschlossen werden. Geräte neuerer Bauart arbeiten meist fehlerfrei, während viele ältere Geräte auf Grund von Normabweichungen nicht verwendbar sind. Ein Gerät sollte also (vor dem Kauf) an der Schnittstelle getestet werden.

Die Maus wird von der SandboxS 2 nur erkannt und verwendet, wenn sie bereits vor dem Einschalten der Spannungsversorgung angesteckt wurde.

Die PS/2-Buchse ist nach Abb. 7.21 am FPGA angeschlossen.

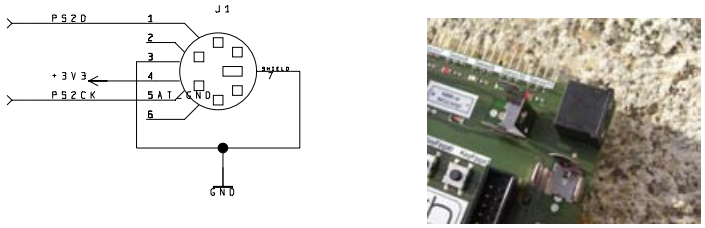


Abbildung 7.21: Anbindung der PS/2-Buchse an das FPGA.

Das Eintreffen eines vollständigen Satzes neuer Werte von der Maus wird mittels des Interrupt-Flags IFPs2 im Interrupt-Flag-Register IrqFlagsOthers (Abschnitt 7.2.1.4 angezeigt. Dieses Interupt-Flag löst einen Interrupt auf Leitung INT0 aus, wenn das Interrupt-Mask-Bit ImRc im Register IrqMaskRcRxCtrl (Abschnitt 7.13.1.1) gesetzt ist (Wert 1).

Da die Werte für XDelta und YDelta den Unterschied zur Position der Maus bei der vorigen Messung angeben, ist es wichtig, alle Wertemeldungen von der Maus sicher zu empfangen. Neue Werte treffen etwa alle 10 ms ein.

7.14.0.4 Register Ps2X

Dieses Register gibt Auskunft über den Weg, den die Maus seit ihrer letzten Ausgabe in X-Richtung zurückgelegt hat.

Bit	7	6	5	4	3	2	1	0
0x87F3	—	—	—	—	—	—	**XOVL**	**XDel.(8)**
0x87F2	**XDelta: signed(7 downto 0)**							

Tabelle 7.23: Register: Ps2X. Nur lesbar.

Bit 9 — XOVL Ist dieses Bit gesetzt (1), so wurde die Maus in X-Richtung zu schnell und weit bewegt, so dass der Weg in X-Richtung nicht mehr in einer 9 Bit breiten Zahl darstellbar ist.

Bits 8 bis 0 — XDelta Diese 9 Bit breite vorzeichenbehaftete Zahl im Zweierkomplementformat gibt den seit der letzten Mausbewegung zurückgelegten Weg in X-Richtung an. Das Vorzeichen gibt die Orientierung der Mausbewegung an.

7.14.0.5 Register Ps2Y

Dieses Register gibt Auskunft über den Weg, den die Maus seit ihrer letzten Ausgabe in Y-Richtung zurückgelegt hat.

Bit	7	6	5	4	3	2	1	0
0x87F5	—	—	—	—	—	—	**YOVL**	**YDel.(8)**
0x87F4	**YDelta: signed(7 downto 0)**							

Tabelle 7.24: Register: Ps2Y. Nur lesbar.

Bit 9 — YOVL Ist dieses Bit gesetzt (1), so wurde die Maus in Y-Richtung zu schnell und weit bewegt, so dass der Weg in Y-Richtung nicht mehr in einer 9 Bit breiten Zahl darstellbar ist.

Bits 8 bis 0 — YDelta Diese 9 Bit breite vorzeichenbehaftete Zahl im Zweierkomplementformat gibt den seit der letzten Mausbewegung zurückgelegten Weg in Y-Richtung an. Das Vorzeichen gibt die Orientierung der Mausbewegung an.

7.14.0.6 Register Ps2Buttons

Dieses Register gibt den Zustand der Maustasten wieder.

Bit	7	6	5	4	3	2	1	0
0x87F7	—	—	—	—	—	—	—	—
0x87F6	—	—	—	—	—	**ButnL**	—	**ButnR**

Tabelle 7.25: Register: Ps2Buttons. Nur lesbar.

Bit 2 — ButnL Bei gedrückter linker Maustaste steht in diesem Bit der Wert 1, sonst der Wert 0.

Bit 1 — ButnR Bei gedrückter rechter Maustaste steht in diesem Bit der Wert 1, sonst der Wert 0.

7.15 Piezosummer mit Tongenerator

In Abb. 7.22 ist der Anschluss des Piezosummers an das FPGA zu sehen. Der Summer ist über einen

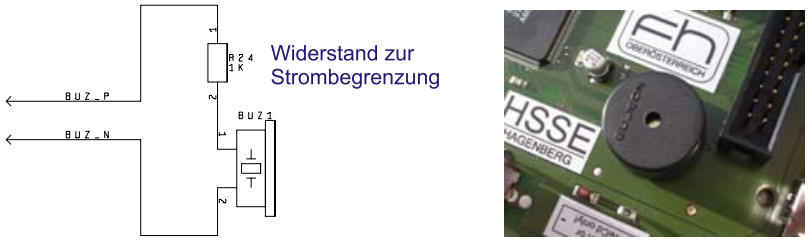


Abbildung 7.22: Piezosummer zur direkten Ansteuerung über das FPGA.

Strombegrenzungswiderstand an zwei Ausgangspins des FPGAs angeschlossen. Eine Strombegrenzung ist auf Grund des kapazitiven Charakters des Piezosummers notwendig. Die erzeugte Lautstärke

hängt bei einem Piezosummer von der angelegten Spannung zwischen den beiden Anschlüssen ab, da sich das Piezokristall entsprechend dieser Spannung verformt. Eine Formänderung wird stets durch eine Spannungsänderung bewirkt. Wird ein Anschluss auf einem festen Pegel gehalten und der andere ändert sich von Low-Pegel zum High-Pegel, dann ist der Unterschied in der Kristallverformung geringer, als wenn sich beide Anschlüsse zugleich auf ihren jeweils entgegengesetzten Wert ändern.

Der Summer wird mit einem Rechtecksignal angesteuert und kann in seiner Tonhöhe und in drei Stufen in seiner Lautstärke eingestellt werden. In geringem Maße ist auch die Klangfarbe beeinflussbar, da ein asymmetrisches Rechtecksignal mit unterschiedlicher High- und Low-Dauer eingestellt werden kann.

7.15.0.7 Register BuzzCtrl

Bit	7	6	5	4	3	2	1	0
0x85FF	—	—	—	—	—	—	—	—
0x85FE	—	—	—	—	—	—	**Volume**	

Tabelle 7.26: Register: BuzzCtrl. Les- und schreibbar.

Bits 1 bis 0 — Volume 00 Aus. Dies ist auch der Initialisierungswert.

01 Mittlere Lautstärke

10 oder 11 Volle Lautstärke

7.15.0.8 Register BuzzPhaseDurationLow

Dieses Register enthält eine vorzeichenlose 16-Bit-Zahl. Diese Zahl gibt an, wie lange die Low-Phase des Rechtecksignales dauert, welches an den Piezosummer angelegt wird. Die Dauer errechnet sich aus dem Produkt der Zeitkonstanten 125 ns mit der im Register angegebenen Zahl.

Bit	7	6	5	4	3	2	1	0
0x85FB	**unsigned(15 downto 8)**							
0x85FA	**unsigned(7 downto 0)**							

Tabelle 7.27: Register: BuzzPhaseDurationLow. Der Initialisierungswert nach dem Einschalten ist 0x2382. Schreib- und lesbar.

7.15.0.9 Register BuzzPhaseDurationHigh

Dieses Register enthält eine vorzeichenlose 16-Bit-Zahl. Diese Zahl gibt an, wie lange die High-Phase des Rechtecksignales dauert, welches an den Piezosummer angelegt wird. Die Dauer errechnet sich aus dem Produkt der Zeitkonstanten 125 ns mit der im Register angegebenen Zahl.

7.16 Erweiterungssteckplatz

In Abb. 7.23 ist der Erweiterungssteckplatz, welcher am FPGA angeschlossen ist dargestellt. Auf dem Steckplatz stehen zur Energieversorgung einer externen Erweiterung sowohl die geregelte Spannung

Bit	7	6	5	4	3	2	1	0
0x85FD	**unsigned(15 downto 8)**							
0x85FC	**unsigned(7 downto 0)**							

Tabelle 7.28: Register: BuzzPhaseDurationHigh. Der Initialisierungswert nach dem Einschalten ist 0x2382. Schreib- und lesbar.

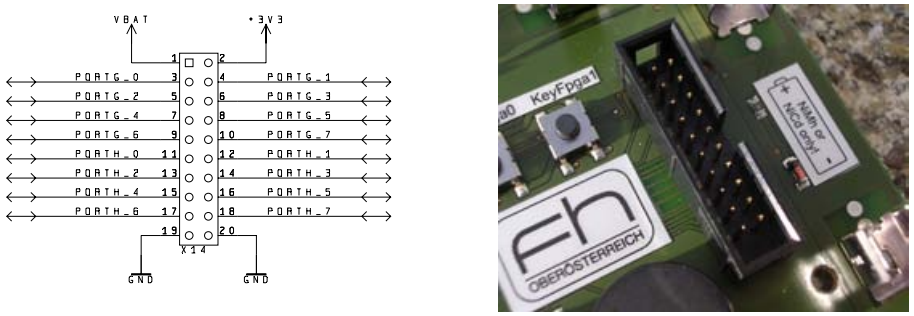


Abbildung 7.23: Als Erweiterungsstecker wird eine 20-polige Stiftwanne eingesetzt.

von 3,3 V, als auch die ungeregelte Akkuspannung zur Verfügung. Die Masseleitung (GND) ist auf zwei Pins ausgeführt. Das FPGA ist über 16 kon gurierbare Pins mit dem Steckplatz verbunden.

In der aktuellen Version des HSSE-Robotics-Chips steht auf dem Erweiterungssteckplatz eine Anschlussmöglichkeit für einen Game-Pad-Controller, wie er für die Sony-Playstation 2 verwendet wird zur Verfügung.

7.17 Sony-Playstation 2 Game-Pad-Controller

Mittels eines einfachen Adapterkabels kann am Erweiterungssteckplatz (siehe Abschnitt 7.16) ein handelsüblicher Game-Pad-Controller für die Sony-Playstation 2 angeschlossen werden. Es kommen

Abbildung 7.24: Beispiel für einen kabellosen Game-Pad-Controller. Ganz links ist auch dessen Gegenstelle zu sehen der in einem Steckverbinder untergebracht ist. Es handelt sich nicht bloß um einen Empfänger, die Funkverbindung ist vielmehr bidirektional.

sowohl kabelgebundene als auch drahtlos (*wireless*) arbeitende Geräte, wie beispielsweise das Gerät in Abb. 7.24 in Frage. In beiden Fällen können die Modi *digital* als auch *analog* verwendet werden. Zwischen diesen wird mittels eines Knopfes mit der Bezeichunng „analog“ auf dem Game-Pad-Controller

umgeschaltet. Der Modus *analog* bietet neben den Funktionen der im Game-Pad-Controller eingebauten Tasten zusätzlich die Ausschläge der beiden Joysticks in Form einer 8 Bit breiten Binärzahl an. Außerdem wird auch der Druck auf den Joystick wie ein Tastendruck übermittelt. Insgesamt stehen 16 Tasten und zwei Joysticks mit jeweils zwei unabhängigen Achsen zur Verfügung. Die weitere Beschreibung geht davon aus, dass der Modus *analog* grundsätzlich aktiviert ist. Ist dies nicht der Fall, so ist die entsprechende Taste auf dem Game-Pad-Controller zu drücken. In Modus *analog* leuchtet normalerweise eine im Controller eingebaute rote LED. Im Modus *digital* ist diese LED hingegen ausgeschaltet.

Game-Pad-Controller für die Sony-Playstation 2 werden nicht nur von Sony, sondern auch von einer großen Zahl weiterer Firmen in hoher Stückzahl hergestellt. Auf dem Markt herrscht ein enormer Preisdruck. Kabellose Versionen sind schon ab 15 € zu bekommen, kabelgebundene Formen liegen preislich meist unterhalb von 10 € teilweise gar bei 5 €. Allerdings gibt es auch entsprechend hohe Qualitätsunterschiede. Viele Game-Pad-Controller an der unteren Preisgrenze lassen es an an Feinfühligkeit fehlen. Vor dem Kauf sollte man also besser ausgiebig testen oder sich an Hand der Erfahrungsberichte von Bekannten oder aus Foren orientieren.

Der attraktive Preis und die praktische Handhabung der Game-Pad-Controller zusammen mit der entsprechenden Auswertungseinheit im HSSE-Robotics-Chip kombinieren sich optimal zur Steuerung von Robotern.

7.17.1 Adapterkabel

Die Steckverbindung der Game-Pad-Controller weist eine eigens von Sony geschaffene Form auf. Diese Steckverbinder sind daher auch nur äußerst schwer zu beschaffen. Für die Bestückung von Platinen sind die Steckverbinder überhaupt nur von Sony selbst zu stark überhöhten Preisen erhältlich. Alternativ hierzu kann ein einfaches Adapterkabel leicht selbst konfektioniert werden. Ausgangspunkt ist ein handlsübliches Verlängerungskabel für Game-Pad-Controller, das zu Preisen zwischen 2 und 6 € erhältlich ist.

Ausgangspunkt: Game-Pad-Verlängerungskabel

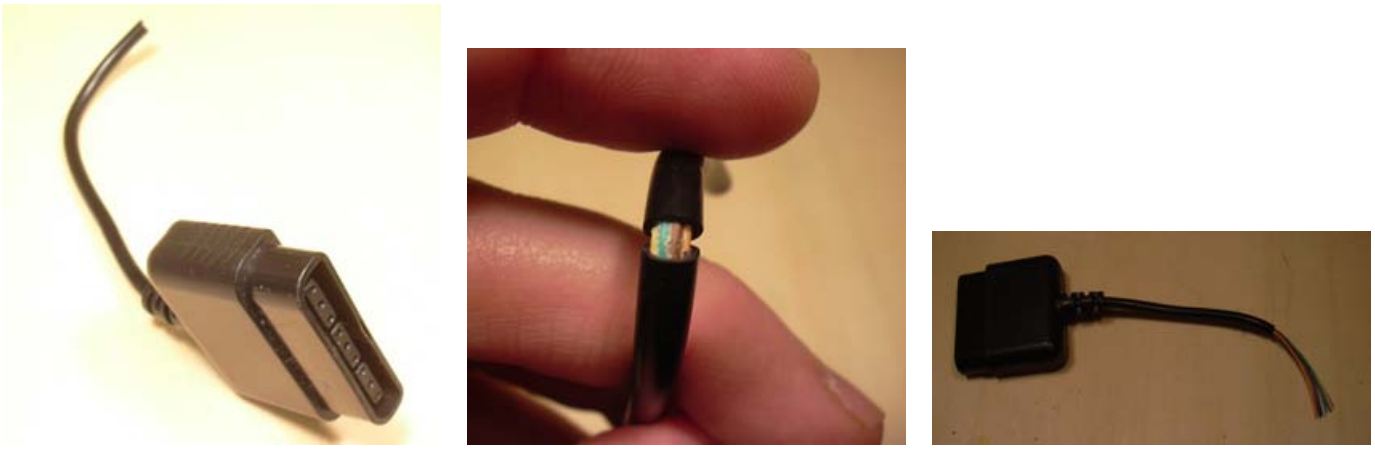


Abbildung 7.25: Aus einem Game-Pad-Verlängerungskabel entsteht das Adapterkabel für die SandboxS 2.

Dieses Verlängerungskabel wird so durchtrennt, das noch etwa 20 cm Kabel am Buchsenteil übrig

bleiben (Abb. 7.25, links).

Nun entfernt man mit einem Messer die Außenhülle des Kabels auf einer Länge von ca. 5 cm (mittleres und rechtes Bild in Abb. 7.25). Hierbei dürfen die Isolierungen der im Innern liegenden Adern des Kabels natürlich nicht angeschnitten werden. Ausreichend Geduld, etwas Vorsicht und eine ruhige Hand werden also vorausgesetzt.

Farbzuordnung

Wenn man auf die Buchse für den Stecker des Game-Pad-Controllers blickt, schaut dies so aus, wie in Abb. 7.26 zu sehen. Die Nummerierung und die Farbzuordnung der einzelnen Adern sind bei einigen

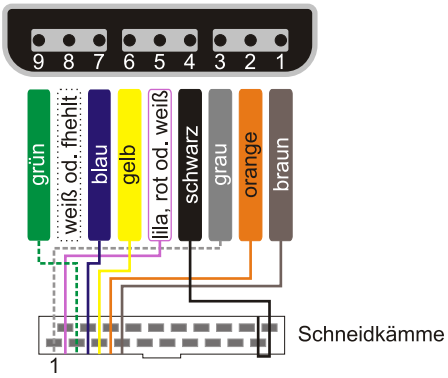


Abbildung 7.26: Nummerierung und gängige Farbzuordnung der einzelnen Kontakte der Buchse des Game-Pad-Verlängerungskabels. Der Kontakt mit der Nummer 8 wird vom Game-Pad-Controller nicht verwendet, so dass meist keine Ader für ihn vorhanden ist. Der Kontakt mit der Nummer 3 dient zur Stromversorgung der Vibrationsmotoren im Game-Pad-Controller. Diese werden für Robotikanwendungen nicht benötigt. Bei drahtlosen Geräten werden die Vibrarionsmotoren von Batterien im Handteil versorgt. Daher kann diese Ader entfallen. Das Signal auf Kontakt Nummer 9 wird von der Schnittstelle des Robotics-Chip nicht benutzt. Auch auf diese Ader kann also auf Wunsch verzichtet werden. Alle anderen Kontakte sind für die Funktion des Game-Pad-Controllers unbedingt notwendig.

Herstellern einheitlich mit der Vorgabe von Sony gehalten. Leider nden sich aber auch vollkommen andere Belegeungen. Es ist also unbedingt nötig, mit einem Multimeter oder einem einfachen Durchgangsprüfer nachzumessen.

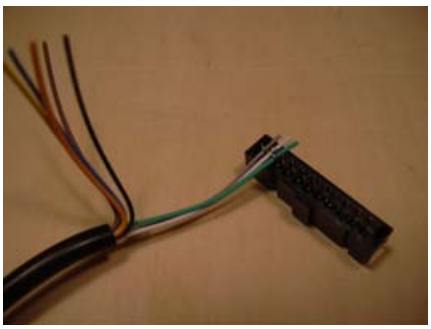
Flachbandkabelsteckverbinder für Einzeladern

Bevor die Einzeladern in den Stecker eingepresst oder besser nur eingedrückt werden können, wird ein spezielles Werkzeug für diesen Zweck hergestellt. Dieses Werkzeug ist eigentlich eines der drei Teile eines Steckers für Flachbandkabel, nämlich der Kabelführungsbügel, der auf das Unterteil des Steckers (das mit den Steckkontakten) aufgedrückt wird. Dieser Kabelführungsbügel enthält halbkreisförmige Hohlkehlen, welche die Einzeladern in die richtige Position für die aus dem Unterteil herausstehenden Schneidklemmverbinder bringen.

Dieses „Werkzeug“ ist eigentlich nur für den einmaligen Gebrauch gedacht und hält im montierten Zustand das Kabel in den Schneidklemmverbindern fest. Hierzu ist es mit zwei Snap-In-Haken versehen, die eine Demontage so gut wie unmöglich machen. Diese Haken werden einfach abgebrochen, wie es Abb. 7.27 zeigt. Allerdings taugt der Bügel dann ntürlich zur Endmontage nichts mehr. Statt des Bügels einen Schraubenzieher oder ähnlich ungeeignetes Gerät zu verwenden hieße dennoch am falschen Platz sparen. Der komplette Stecker kostet ohnehin nur wenige Cent.

Abbildung 7.27: Herstellung des Montagewerkzeugs für Einzeladern durch Abbrechen der Snap-In-Haken.

Nun können die einzelnen Adern des Kabels eine nach der anderen in die Schneidklemmverbinder eingedrückt werden. Abb. 7.29 zeigt einen Zwischenschritt. Die Adern des Kabels werden einzeln in



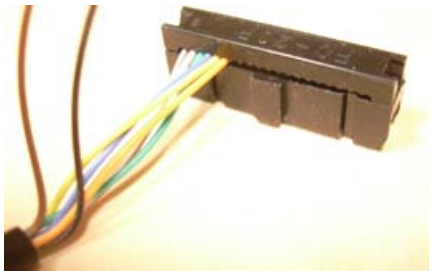


Abbildung 7.28: Einzeladern werden in den eigentlich für Flachbandkabel gedachten Stecker eingelegt. Jede Ader wird in den zugehörigen Schneidkamm eingedrückt, wobei ein Teil eines weiteren Steckers behilflich ist.

einen Wannenstecker eingelegt, der vom Hersteller ursprünglich für Flachbandkabel gedacht ist. Dies geht mit einem einfachen Trick recht leicht, wozu zwei solche Stecker benötigt werden: Einer der beiden dient lediglich als Werkzeug beim Einlegen der einzelnen Adern während der andere seiner ursprünglich zugedachten Aufgabe als Steckverbinder nachgehen darf. Die Farbzuordnung wird wieder Abb. 7.26 entnommen.

Sollte das verwendete Kabel andere als die dargestellten Farben verwenden, wird dies am besten zuerst mittels eines Durchgangsprüfers gemessen und im Bild7.26 eingetragen. Ein fehlerhaftes Kabel kann zur Zerstörung der SandboxS 2 oder des Game-Pad-Controllers führen.

Bevor ein Kabelführungsbügel dessen Snap-In-Halter nicht abgebrochen wurden entgültig alle Adern sichert, sollte nochmals mit dem Durchgangsprüfer die korrekte Zuordnung der Kontakte im Sony-Steckverbinder zu den noch freiligenden Schneidklemmkontakten geprüft werden. Wurde eine falsche Ader eingedrückt oder Adern verwechselt, so ist dieser Fehler zu diesem Zeitpunkt noch leicht zu beheben. Bei wiederholter Montage muss aber wieder ein unverletztes Stück der Kabelisolation auf dem Schneidklemmverbinder zu liegen kommen!

Zugentlastung des Kabels

Die Zugentlastung wird in der Abfolge aus Abb. 7.30 unter Zuhilfenahme zweier Kabelbinder vorgenommen. Zunächst wird der mit dem Flachbandsteckverbinder gelieferte Umlenkbügel aufgesteckt und festgepresst. Anschließend wird das Kabel mit zwei Kabelbindern an diesem Bügel befestigt. Das Endergebnis ist im rechten Foto zu sehen.

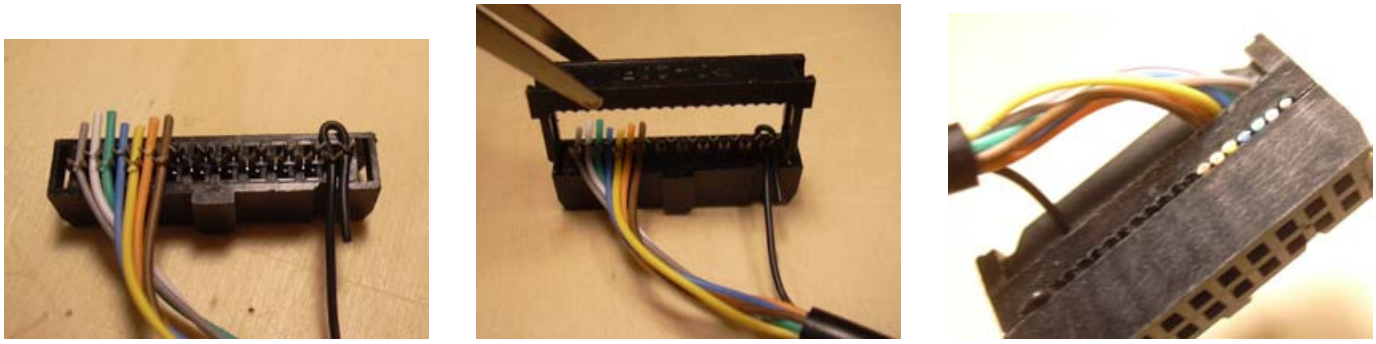


Abbildung 7.29: Einzeladern werden in den eigentlich für Flachbandkabel gedachten Stecker eingelegt. Jede Ader wird in den zugehörigen Schneidkamm eingedrückt, wobei ein Teil eines weiteren Steckers behilflich ist.

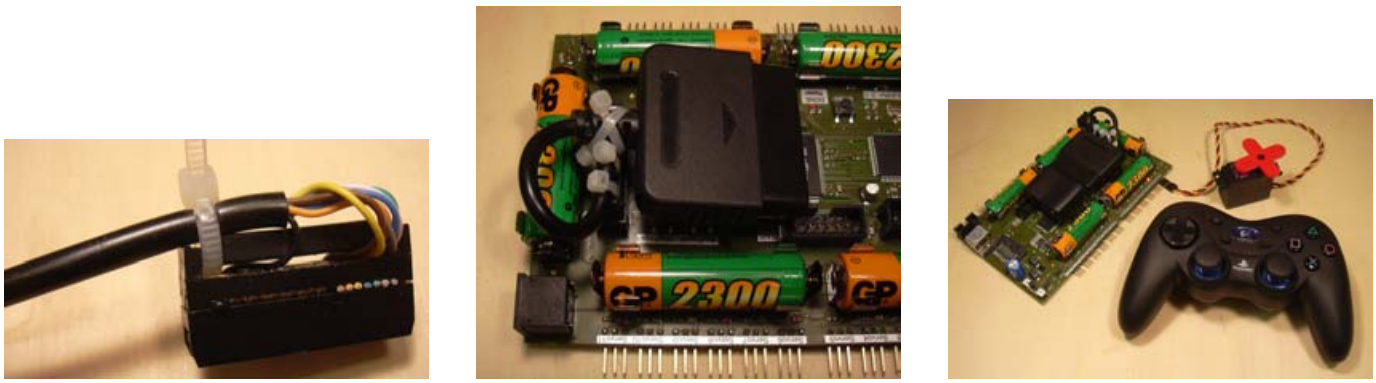


Abbildung 7.30: Zugentlastung mit einem Umlenkbügel und zwei Kabelbindern. Im rechten Bild ist das Komplettsystem bei Verwendung eines Playstation-2-Controllers der Firma Logitech zu sehen.

7.17.1.1 Register SonyPs2GamePadButtons

Dieses Register gibt an, welche Tasten auf dem Game-Pad-Controller gedrückt sind und welche nicht. Die nachfolgende Registerbeschreibung gilt nur dann vollständig, wenn der Game-Pad-Controller mit der Taste „analog“ in den Modus *analog* (rote LED leuchtet) versetzt wurde! Ist der Modus *digital* ausgewählt, so sind die Bits mit der Bezeichnung JoyR und JoyL fest mit dem Wert 1 belegt. Jedes

Bit	7	6	5	4	3	2	1	0
0x87C3	□	×	○	△	R1	L1	R2	L2
0x87C2	◄	▼	►	▲	Start	JoyR	JoyL	Select

Tabelle 7.29: Register: SonyPs2GamePadButtons. Nur lesbar. Die Zeichen □, ×, ○ und △ sind als Sinnbilder auf den Tasten auf der rechten Seite des Controllers zu nden. Die ausgefüllten Pfeile bezeichnen die Richtungen, in welche der „Tastenjoystick“ auf der linken Seite des Controllers betätigt wird. Die beiden Tasten „Start“ und „Select“ nden sich in der Mitte. Die Tasten mit den Bezeichnungen L1, R1, L2 und R2 be nden sich auf den „Schultern“ des Controllers, wo sie bequem mit den Zeige ngern erreichbar sind. Die Bits mit den Namen JoyR und JoyL geben an, ob der jeweilige Joystick niedergedrückt wird.

Bit im Register entspricht einer Taste am Game-Pad-Controller. Ist die entsprechende Taste gedrückt, so wird für das Bit der Wert 0 gelesen, ansonsten der Wert 1 . Wird keine Taste gedrückt, so steht entsprechend in allen Bits dieses Registers der Wert 1 .

7.17.1.2 Register SonyPs2GamePadJoyR

Dieses Register gibt die Stellung an, in der sich der rechte Joystick be ndet. Die Auslenkung des Joysticks wird durch zwei vorzeichenlose 8 Bit breite Binärzahlen angegeben, wobei jedoch meist gar nicht mit der vollen Genauigkeit einer 8-Bit-Zahl gerechnet werden kann. Insbesondere billige Controller verfügen meist über weit geringere Auflösungen.

Die Daten sind in diesem Register nur verfügbar, wenn der Game-Pad-Controller mit der Taste „analog“ in den Modus *analog* (rote LED leuchtet) versetzt wurde! Im Modus *digital* ist das Register komplett mit Bits des Wertes 1 belegt.

Bit	7	6	5	4	3	2	1	0
0x87C5	**JoyR – y-Achse: unsigned(7 downto 0)**							
0x87C4	**JoyR – x-Achse: unsigned(7 downto 0)**							

Tabelle 7.30: Register: SonyPs2GamePadJoyR. Nur lesbar.

Bits 15 bis 8 — JoyR – y-Achse Dieses Register enthält die momentane Auslenkung für die y-Achse des rechten Joysticks ausgedrückt durch eine vorzeichenlose 8 Bit breite Zahl, wobei 0x00 der Auslenkung nach ganz oben und 0xFF der Auslenkung nach ganz unten entspricht.

Bits 7 bis 0 — JoyR – x-Achse Dieses Register enthält die momentane Auslenkung für die x-Achse des rechten Joysticks ausgedrückt durch eine vorzeichenlose 8 Bit breite Zahl, wobei 0x00 der Auslenkung nach ganz links und 0xFF der Auslenkung nach ganz rechts entspricht.

7.17.1.3 Register SonyPs2GamePadJoyL

Dieses Register gibt die Stellung des linken Joysticks in der gleichen Weise, wie schon beim Register SonyPs2GamePadJoyR für den rechten Joystick beschrieben an.

Bit	7	6	5	4	3	2	1	0
0x87C7	**JoyL – y-Achse: unsigned(7 downto 0)**							
0x87C6	**JoyL – x-Achse: unsigned(7 downto 0)**							

Tabelle 7.31: Register: SonyPs2GamePadJoyL. Nur lesbar.

Bits 15 bis 8 — JoyL – y-Achse Dieses Register enthält die momentane Auslenkung für die y-Achse des linken Joysticks ausgedrückt durch eine vorzeichenlose 8 Bit breite Zahl, wobei 0x00 der Auslenkung nach ganz oben und 0xFF der Auslenkung nach ganz unten entspricht.

Bits 7 bis 0 — JoyL – x-Achse Dieses Register enthält die momentane Auslenkung für die x-Achse des linken Joysticks ausgedrückt durch eine vorzeichenlose 8 Bit breite Zahl, wobei 0x00 der Auslenkung nach ganz links und 0xFF der Auslenkung nach ganz rechts entspricht.

Das Interface für das Playstation-Gamepad verfügt nicht über die Möglichkeit, neue Werte mittels eines Interrupt-Flags anzuzeigen. Die Abfrage des Gamepads erfolgt im gleichen Rhytmus, in dem auch die Servopulsausgabe erfolgt (also ca. alle 20 ms). Daher kann ein Interrupt für das Gamepad erzeugt werden, indem ein Servosequencer mit dem Wert NrOfDeltas=1 programmiert wird. Dieser Sequencer löst dann alle 20 ms ein Interrupt-Flag aus. Sollen die Werte vom GamePad seltener eingelesen werden, so kann NrOfDeltas entsprechend auf eine höhere Zahl gesetzt werden.

7.18 Globaler Timer

Der Globale Timer zählt beginnend mit dem Wert 0 die PWM-Pulse, die vom Servopulsgenerator erzeugt werden. Unmittelbar nach dem Einschalten ist dessen Pulslänge auf ca. 20 ms eingestellt. Je nach Einstellung der Register TuneFine und TuneCoarse laut Abschnitt 7.7.1 kann die Pulslänge variieren.

7.18.0.4 Register GlobalTimer

Dieses Register enthält eine vorzeichenlose 16-Bit-Zahl. Diese Zahl gibt an, wie lange die Stromversorgung des Robotics-Chips bereits eingeschaltet ist. Die Dauer errechnet sich aus dem Produkt der Zeitkonstanten welche durch die beiden Register TuneFine und TuneCoarse (siehe Abschnitt 7.7.1) eingestellt wird (nach dem Einschalten 20 ms) mit der im Register angegebenen Zahl.

Bit	7	6	5	4	3	2	1	0
0x87FF	**unsigned(15 downto 8)**							
0x87FE	**unsigned(7 downto 0)**							

Tabelle 7.32: Register: GobalTimer. Der Initialisierungswert nach dem Einschalten ist 0x0000. Nur lesbar.

7.19 Versionsnummer des Robotics-Chips

Die Versionsnummer des Robotics-Chips ist als Nur-Lese-Register angelegt, das per Software erfragt werden kann. Die Version ist als vierstellige Dezimalzahl in ASCII-Codierung, also als String mit vier Zeichen verfügbar.

7.19.0.5 Register VersionDigits32

Bit	7	6	5	4	3	2	1	0
0x87F9	**ASCII-Zeichen: 1000er-Stelle**							
0x87F8	**ASCII-Zeichen: 100er-Stelle**							

Tabelle 7.33: Register: VersionDigits32. 1000er- und 100er-Stelle der Versionsnummer des Robotics-Chips. Nur lesbar.

7.19.0.6 Register VersionDigits10

Bit	7	6	5	4	3	2	1	0
0x87FB	**ASCII-Zeichen: 10er-Stelle**							
0x87FA	**ASCII-Zeichen: 1er-Stelle**							

Tabelle 7.34: Register: VersionDigits32. 10er- und 1er-Stelle der Versionsnummer des Robotics-Chips. Nur lesbar.

7.20 Demo-Mode

Der Robotics-Chip wird aus Gründen des Kopierschutzes 10 bis 11 Minuten nach dem Einschalten deaktiviert. Der deaktivierte Zustand wird durch Aufleuchten beider Message-LED am FPGA (MsgFpga0 und MsgFpga1) angezeigt. Außerdem wird eine mittlere Lautstärke des Summers eingestellt.

7.21 RAM-Erweiterung

Vom FPGA werden 2 kByte RAM der Wortbreite 8 Bit in den Adressbereich des AVRs eingeblendet. Diese be nden sich im Adressbereich 0x9000 – 0x97FF. Dieses externe RAM kann als Erweiterung des internen RAMs verwendet werden, wobei jedoch im Unterschied zu diesem pro RAM-Zugriff drei Taktzyklen statt eines einzigen notwendig sind.

Das RAM wird nach dem Einschalten weitgehend mit dem Wert 0x00 initialisiert. Am Anfang des RAM-Bereichs kann eine Copyright-Message ausgelesen werden.

Um gegenüber zukünftigen Veränderungen im Funktionsumfang des HSSE-Robotics-Chip flexibel zu sein, wird *dringend* empfohlen, den Softwareentwurf keinesfalls auf die vollständige Verfügbarkeit der 2 kByte RAM zu stützen.

7.22 CORDIC-Koprozessor

In der Robotik spielen trigonometrische Funktionen eine wichtige Rolle etwa bei der Transformation von Koordinatensystemen oder der Berechnung der inversen Kinematik eines Roboters.

Zur Berechnung der Werte trigonometrischer Funktionen gibt es unterschiedliche Methoden, die alle auf einer schrittweisen Annäherung an den gesuchten Funktionswert beruhen. Jeder Schritt verbessert die Genauigkeit der Lösung. Innerhalb eines solchen Schrittes sind zwar ebenfalls mathematische Berechnungen notwendig, diese sind aber relativ einfacher Natur; meist handelt es sich um Additionen und Multiplikationen.

Die bekannten Methoden unterscheiden sich stark im Aufwand der für jeden Rechenschritt notwendig ist und auch in der Anzahl der Schritte, die zur Erreichung einer gewünschten Genauigkeit erforderlich sind. Beide Eigenschaften zusammen entscheiden über die Eignung einer Berechnungsmethode zur Umsetzung in Hardware oder auch Software. Die bekannteste Methode zur Berechnung von Funktionen wie $\cos\alpha$ oder $\arctan\gamma$ ist sicher die so genannte Taylor-Reihe. Eine Taylor-Reihe ist für die Umsetzung in Software oder Hardware zwar geeignet, erfordert aber einen recht hohen Berechnungsaufwand, da einige Divisionen auszuführen sind. In der Praxis der Computerarithmetik konnte dieses Verfahren nie Fuß fassen, denn es gibt seit den 50er Jahren des 20. Jahrhunderts einen wesentlich ef zienteren Konkurrenten — den CORDIC-Algorithmus (VOLDER (1959), WALTHER (1971)).

Der CORDIC-Algorithmus führt die Berechnung vielfältiger elementarer Funktionen auf Berechnungen zurück, die in Hardware sehr ef zient ausführbar sind: Schiebeoperationen, Additionen und das Nachschlagen einiger Tabellenwerte.

CORDIC-Auftragsbearbeitung

Der HSSE-Robotics-Chip muss aus Platzgründen mit einer einzigen CORDIC-Recheneinheit auskommen. Damit diese über einen möglichst langen Zeitraum hinweg unabhängig vom AVR-Mikrocontroller ihre Berechnungen ausführen kann, ist ein Auftragssystem implementiert. Dieses kann mit mehreren Berechnungen beauftragt werden und unterbricht den Mikrocontroller erst nach Bearbeitung des kompletten Auftrags mittels eines Interrupt-Requests. Alternativ kann der Mikrocontroller auch Teilergebnisse erfragen und bereits nach deren Fertigstellung den entsprechenden Teil eines neuen Auftrags notieren.

Eine Berechnung des CORDIC-Prozessors besitzt drei Eingabewerte x, y und z sowie zwei Parameter, welche die Rechenvorschrift (trigonometrisch, linear, hyperbolisch) und den CORDIC-Modus (*Rotate*, *Vectoring*) auswählen. Diese Eingabewerte, werden vom Mikrocontroller aus in einem Registerblock abgelegt, der sich aus vier 16 Bit breiten Registern zusammensetzt.

Die Zahlenformate der drei Eingabewerte x, y und z sind dabei sowohl vom Modus als auch vom Koordinatensystem abhängig und sind in Abschnitt 7.22.1 aufgelistet. Die Rechenvorschrift wird wie folgt bestimmt:

Bit	7	6	5	4	3	2	1	0
Adr+1	—	—	—	—	—	—	—	—
Adr	—	—	—	—	—	**CoordSystem**		**Mode**

Tabelle 7.35: Register: Koordinatensystem und Modus eines CORDIC-Berechnungsauftrages.

Bit 2 – 1 — CoordSystem Koordinatensystem der CORDIC-Berechnung. Die Zuordnung von Bits und Koordinatensystem ist in Tabelle 7.36 ersichtlich.

Koordinatensystem	CoordSystem
linear	00
zirkular	01
hyperbolisch	10

Tabelle 7.36: Binärwerte zur Auswahl des Koordinatensystems

Bit 0 — Mode Modus der CORDIC-Berechnung. Der Rotationsmodus wird ausgewählt, wenn dieses Bit auf 0 gesetzt wird, bei 1 wird der Vektormodus verwendet.

Tabelle 7.37 gibt eine Übersicht, welche Funktionen mit welchem Koordinatensystem bzw. Modus berechnet werden können.

Im Adressbereich des CORDIC-Coprozessor können bis zu 32 solcher Eingabepakete hintereinander abgelegt werden. Es ist möglich, jede beliebige Anzahl solcher Pakete zwischen 1 und 32 zu einen Auftrag zusammen zu schnüren.

Ist der Mikrocontroller mit dem Ablegen der Eingabewerte in den entsprechenden Registern fertig, so schreibt er in das Register NrOfCordicToDo die Anzahl der Pakete, die berechnet werden sollen. Damit sind automatisch immer die Pakete mit den Nummern 1 bis Anzahl(Pakete) innerhalb des Adressbereiches gemeint. Durch das Schreiben dieses Registers wird die Berechnungstätigkeit der CORDIC-Recheneinheit gestartet. Diese beginnt beim Paket mit der höchsten Nummer und arbeitet sich Paket für Paket bis zum Paket Nr. 1 weiter. Das Register NrOfCordicToDo dient synchron zum Berechnungsvorgang zur Buchführung und enthält stets die Anzahl der noch ausstehenden Pakete.

Alle in Auftrag gegebenen Pakete mit einer höheren Nummer, als sie aus dem Register NrOfCordicToDo gelesen wird, sind bereits fertig berechnet. Die Ergebnisse legt der CORDIC-Prozessor in den gleichen Registern ab, in denen er die Eingaben vorgefunden hat. Der Mikrocontroller kann diese Ergebnisse bereits auslesen, während der CORDIC-Prozessor noch mit Paketen mit niedrigerer Nummer beschäftigt ist.

Um den CORDIC-Prozessor weitgehend auszulasten, ist es möglich, unmittelbar nach dem Auslesen der Ergebnisse aus einem Paket bereits wieder die Daten für einen neuen Berechnungsauftrag in das Registerpaket zu packen. Der neue Auftrag sollte aber erst nach der vollständigen Abarbeitung aller noch zu erledigenden Berechnungen durch das Schreiben auf NrOfCordicToDo erteilt werden. Die vollständige Bearbeitung des letzten Auftrags ist sowohl am Wert 0 in NrOfCordicToDo als auch am Wert des Interrupt-Flags IFCord (siehe Abschnitt 7.2.1.4) erkennbar. Dieses Interrupt-Flag wird gesetzt, sobald der Wert von NrOfCordicToDo von 1 nach 0 übergeht.

Das Interrupt-Flag IFCord kann darüber hinaus auch einen Interrupt-Request auf der Leitung INT0 auslösen, um dem Mikrocontroller die Fertigstellung des Auftrages mitzuteilen. Ein Interrupt wird vom Interrupt-Flag ausgelöst, wenn das Bit ImCord im Register IrqMaskRcRxCtrl (Abschnitt 7.13.1.1) den Wert 1 trägt. Hat es den Wert 0 , so wird die Auslösung eines Interrupt-Requests unterbunden.

Ein- und Ausgabewerte des CORDIC-Algorithmus

Version	Rotationsmodus $(d_i = \text{sign}(z_i),\ z_n \rightarrow 0)$	Vektormodus $(d_i = -\text{sign}(x_i y_i),\ y_n \rightarrow 0)$
zirkular	x, y, z → CORDIC → $K_{cir}(x\cos z - y\sin z)$, $K_{cir}(y\cos z + x\sin z)$, 0 $\sin(a),\ \cos(a)$ $A\angle\Phi \rightarrow a + jb$	x, y, z → CORDIC → $K_{cir}\sqrt{x^2+y^2}$, 0, $z + \arctan\frac{y}{x}$ $\arctan(a),\ \sqrt{a^2+b^2}$ $a + jb \rightarrow A\angle\Phi$
linear	x, y, z → CORDIC → x, $y + xz$, 0 $x \cdot z$	x, y, z → CORDIC → x, 0, $z + \frac{y}{x}$ $\frac{y}{x}$
hyperbolisch	x, y, z → CORDIC → $K_{hyp}(x\cosh z - y\sinh z)$, $K_{hyp}(y\cosh z + x\sinh z)$, 0 $\sinh(a),\ \cosh(a)$	x, y, z → CORDIC → $K_{hyp}\sqrt{x^2-y^2}$, 0, $z + \text{artanh}\frac{y}{x}$ $\text{artanh}(a),\ \sqrt{a^2-b^2}$

Tabelle 7.37: Verschiedene Modi und Koordinatensysteme (KREUZGRUBER (2007))

Die Berechnungen weisen je nach dem gewählten Koordinatensystem eine bestimmte Verstärkung auf (siehe Tabelle 7.38):

Modus	zirkular	linear	hyperbolisch
K	1,646760258121	1	0,8281593609602

Tabelle 7.38: K in Abhängigkeit vom Modus (KREUZGRUBER (2007))

7.22.1 Zahlenformate, Genauigkeit

Der CORDIC-Algorithmus benötigt drei 16-Bit Register für die Ein-/Ausgangswerte, deren Datenformate abhängig vom gewählten Koordinatensystem und Modus interpretiert werden. In diesem Abschnitt sind alle Datenformate jeweils für die Eingangs- und Ausgangswerte von x, y und z angeführt.

Da in jeder Berechnung ein Wert zu 0 wird — z im Rotationsmodus bzw. y im Vektormodus — ist für diese Ausgangswerte kein Zahlenformat angegeben. Ob eine Berechnung konvergiert hat (erfolgreich war) kann durch lesen dieser Register bestimmt werden. Theoretisch sollten diese Register immer gleich 0 werden, wenn die Berechnung konvergiert. Durch die nicht unendliche Anzahl von Berechnungsschritten und die beschränkte Darstellung der Zahlenwerte wird dieser Wert häu g nicht exakt 0. Überschreitet der Betrag dieser „Ungenauigkeit“ den Wert 20, so kann davon ausgegangen werden, dass die Berechnung nicht erfolgreich war und alle Ergebnisse ungültig sind.
Die Überprüfung dieses Registers ist nicht nötig, wenn sichergestellt wird, dass die angegebenen Wertebereiche nicht überschritten werden.

Zirkularer Rotationsmodus

- z entspricht in diesem Modus einem Winkel in Radiant wobei π dem Wert 12868 ($\widehat{=}\ \pi \cdot 2^{12}$) entspricht. Um sicherzustellen, dass die Berechnung konvergiert, muss der Winkel innerhalb des Bereichs $0 – \pi \cdot 2^{12}$ liegen.
- Sowohl das Ergebnis von Sinus und Cosinus weisen eine Verstärkung von K_{cir} auf.
- Die Genauigkeit von beiden Berechnungen beträgt 14,54 Bit.

Die Datenformate in diesem Modus sind in den Abbildungen 7.39 und 7.40 dargestellt.

Bit	7	6	5	4	3	2	1	0
Adr+1	**unsigned(3 downto -4)**							
Adr	**unsigned(-5 downto -12)**							

Tabelle 7.39: Register: Eingabewerte von x, y und z des CORDIC-Algorithmus (Zirkularer Rotationsmodus). Schreib- und lesbar.

Bit	7	6	5	4	3	2	1	0
Adr+1	**signed(1 downto -6)**							
Adr	**signed(-7 downto -14)**							

Tabelle 7.40: Register: Ausgabewerte von x und y des CORDIC-Algorithmus (Zirkularer Rotationsmodus). Schreib- und lesbar.

Zirkularer Vektormodus

- Der Eingangwert von z sollte in diesem Modus immer 0 sein.
- Nur das Ergebnis der Wurzelfunktion weist eine Verstärkung von K_{cir} auf. Die Berechnung des Arkustangens hat keine Verstärkung!
- Die Genauigkeit der Berechnung des Arkustangens beträgt 14,7 Bit, jene der Wurzelfunktion 16,15 Bit.

Die Datenformate in diesem Modus sind in den Abbildungen 7.41 und 7.42 dargestellt.

Linearer Rotationsmodus

Die Genauigkeit der Berechnung beträgt 17,19 Bit.
Die Datenformate in diesem Modus sind in den Abbildungen 7.43, 7.44 und 7.45 dargestellt.

Bit	7	6	5	4	3	2	1	0
Adr+1	**unsigned(1 downto -6)**							
Adr	**unsigned(-7 downto -14)**							

Tabelle 7.41: Register: Eingabewerte von x und y des CORDIC-Algorithmus (Zirkularer Vektormodus). Schreib- und lesbar.

Bit	7	6	5	4	3	2	1	0
Adr+1	**unsigned(1 downto -6)**							
Adr	**unsigned(-7 downto -14)**							

Tabelle 7.42: Register: Ausgabewerte von x und z CORDIC (Zirkularer Vektormodus). Schreib- und lesbar.

Linearer Vektormodus

- Der Eingangwert von z sollte in diesem Modus immer 0 sein und y muss immer kleiner bzw. gleich gross sein wie x.
- Die Genauigkeit der Berechnung beträgt 18,36 Bit.

Die Datenformate in diesem Modus sind in den Abbildungen 7.46, 7.47 und 7.48 dargestellt.

Hyperbolischer Rotationsmodus

Dieser Modus wird nicht unterstützt.

Hyperbolischer Vektormodus

- Die Genauigkeit der Berechnung der Wurzelfunktion beträgt 15,94 Bit.
- Die Berechnung des Arkustangens-Hyperbolicus wird nicht unterstützt.
- Für die Berechnung der Wurzelfunktion von a muss $x = a + \frac{1}{4}$ und $y = a - \frac{1}{4}$ gewählt werden, wobei sich a innerhalb des Wertebereichs 0.03 – 1 be nden muss.
- Der Eingangwert von z sollte in diesem Modus immer 0 sein.

Die Datenformate in diesem Modus sind in den Abbildungen 7.49 und 7.50 dargestellt.

Bit	7	6	5	4	3	2	1	0
Adr+1	**unsigned(15 downto 8)**							
Adr	**unsigned(7 downto 0)**							

Tabelle 7.43: Register: Eingabewerte von x, y und z des CORDIC-Algorithmus (Linearer Rotationsmodus). Schreib- und lesbar.

Bit	7	6	5	4	3	2	1	0
Adr+1	**unsigned(3 downto -4)**							
Adr	**unsigned(-5 downto -12)**							

Tabelle 7.44: Register: Ausgabewert von x des CORDIC-Algorithmus (Linearer Rotationsmodus). Schreib- und lesbar.

Bit	7	6	5	4	3	2	1	0
Adr+1	**unsigned(15 downto 8)**							
Adr	**unsigned(7 downto 0)**							

Tabelle 7.45: Register: Ausgabewert von y des CORDIC-Algorithmus (Linearer Rotationsmodus). Schreib- und lesbar.

Bit	7	6	5	4	3	2	1	0
Adr+1	**unsigned(15 downto 8)**							
Adr	**unsigned(7 downto 0)**							

Tabelle 7.46: Register: Eingabewerte von x und y des CORDIC-Algorithmus (Linearer Vektormodus). Schreib- und lesbar.

Bit	7	6	5	4	3	2	1	0
Adr+1	**unsigned(9 downto 2)**							
Adr	**unsigned(1 downto -6)**							

Tabelle 7.47: Register: Ausgabewert von x des CORDIC-Algorithmus (Linearer Vektormodus). Schreib- und lesbar.

Bit	7	6	5	4	3	2	1	0
Adr+1	**unsigned(5 downto -2)**							
Adr	**unsigned(-3 downto -10)**							

Tabelle 7.48: Register: Ausgabewert von z des CORDIC-Algorithmus (Linearer Vektormodus). Schreib- und lesbar.

Bit	7	6	5	4	3	2	1	0
Adr+1	**signed(1 downto -6)**							
Adr	**signed(-7 downto -14)**							

Tabelle 7.49: Register: Eingabewerte von x und y des CORDIC-Algorithmus. Schreib- und lesbar.

Bit	7	6	5	4	3	2	1	0
Adr+1	**signed(-1 downto -8)**							
Adr	**signed(-9 downto -16)**							

Tabelle 7.50: Register: Ausgabewert von x des CORDIC-Algorithmus. Schreib- und lesbar.

7.23 Adressraum im Überblick

Tabelle 7.51: Überblick über den Registeraddressraum. n_S ist die Servo- und n_L die LED-Nummer (jeweils von 0 bis 23). Der jeweils verfügbare Adressbereich ist nicht für 24, sondern bis zu 32 Servos bzw. LEDs reserviert.

Adr(AVR)	Ende	Words	Beschreibung
16-Bit-Register – AVR kann schreiben und lesen			
0x8000+2n_S	0x803F	32	Phi2PwmTable(0) Servo n_S
0x8040+2n_S	0x807F	32	Phi2PwmTable(1) Servo n_S
0x8080+2n_S	0x80BF	32	Phi2PwmTable(2) Servo n_S
0x80C0+2n_S	0x80FF	32	Phi2PwmTable(3) Servo n_S
0x8100+2n_S	0x813F	32	Phi2PwmTable(4) Servo n_S
0x8140+2n_S	0x817F	32	Phi2PwmTable(5) Servo n_S
0x8180+2n_S	0x81BF	32	Phi2PwmTable(6) Servo n_S
0x81C0+2n_S	0x81FF	32	Phi2PwmTable(7) Servo n_S
0x8200+2n_S	0x823F	32	Phi2PwmTable(8) Servo n_S
0x8240+2n_S	0x827F	32	Phi2PwmTable(9) Servo n_S
0x8280+2n_S	0x82BF	32	Phi2PwmTable(10) Servo n_S
0x82C0+2n_S	0x82FF	32	Phi2PwmTable(11) Servo n_S
0x8300+2n_S	0x833F	32	Phi2PwmTable(12) Servo n_S
0x8340+2n_S	0x837F	32	Phi2PwmTable(13) Servo n_S
0x8380+2n_S	0x83BF	32	Phi2PwmTable(14) Servo n_S
0x83C0+2n_S	0x83FF	32	Phi2PwmTable(15) Servo n_S
0x8400+2n_S	0x843F	32	Phi2PwmTable(16) Servo n_S
0x8440+2n_S	0x847F	32	ServoMode/IRQ-Mask Servo n_S
0x8480+2n_S	0x84BF	32	PwmLen/PhiStart Servo n_S
0x84C0+2n_S	0x84FF	32	DeltaPhi Servo n_S
0x8500+2n_S	0x853F	32	NrOfDeltas Servo n_S
0x8540+2n_S	0x857F	32	LedBrightness LED n_L
0x8580	0x85F1	57	reserviert
0x85F2	0x85F3	1	intern
0x85F4	0x85F5	1	IrqMaskRcRxCtrl
0x85F6	0x85F7	1	TuneFine
0x85F8	0x85F9	1	TuneCoarse
0x85FA	0x85FB	1	BuzzPhaseDurationLow
0x85FC	0x85FD	1	BuzzPhaseDurationHigh
0x85FE	0x85FF	1	BuzzCtrl
16-Bit-Register – AVR kann nur lesen			
0x8600+2n_S	0x863F	32	DeltasToGo Servo n_S
0x8640+2n_S	0x867F	32	Current DeltaPhi Servo n_S
0x8680+2n_S	0x86BF	32	Current Phi Servo n_S
0x86C0+2n_S	0x86FF	32	Current PWM Servo n_S
0x8700	0x87BD	95	reserviert

Adr(AVR)	Ende	Words	Beschreibung
0x87BE	0x87BF	1	intern
0x87C0	0x87C1	1	intern
0x87C2	0x87C3	1	SonyPs2GamePadButtons
0x87C4	0x87C5	1	SonyPs2GamePadJoyR
0x87C6	0x87C7	1	SonyPs2GamePadJoyL
0x87C8	0x87D7	8	reserviert
0x87D8	0x87D9	1	RcRxChannel(1)
0x87DA	0x87DB	1	RcRxChannel(2)
0x87DC	0x87DD	1	RcRxChannel(3)
0x87DE	0x87DF	1	RcRxChannel(4)
0x87E0	0x87E1	1	RcRxChannel(5)
0x87E2	0x87E3	1	RcRxChannel(6)
0x87E4	0x87E5	1	RcRxChannel(7)
0x87E6	0x87E7	1	RcRxChannel(8)
0x87E8	0x87E9	1	RcRxChannel(9)
0x87EA	0x87EB	1	RcRxChannel(10)
0x87EC	0x87ED	1	RcRxChannel(11)
0x87EE	0x87EF	1	RcRxChannel(12)
0x87F0	0x87F1	1	RcRxStatus
0x87F2	0x87F3	1	Ps2X
0x87F4	0x87F5	1	Ps2Y
0x87F6	0x87F7	1	Ps2Buttons
0x87F8	0x87FB	2	Version
0x87FC	0x87FD	1	reserviert
0x87FE	0x87FF	1	GlobalTimer
16 Bit-Register – AVR kann schreiben und lesen			
0x8800	0x8801	1	NrOfCordicToDo
0x8802	0x8807	3	reserviert
0x8808+$2n_C$	0x8809+$2n_C$	31	Ein- und Ausgabewert x CORDIC-Block Nr. n_C
0x880A+$2n_C$	0x880B+$2n_C$	31	Ein- und Ausgabewert y CORDIC-Block Nr. n_C
0x880C+$2n_C$	0x880D+$2n_C$	31	Ein- und Ausgabewert z CORDIC-Block Nr. n_C
0x880E+$2n_C$	0x880F+$2n_C$	31	*CoordSystem* und *Mode* CORDIC-Block Nr. n_C
0x8900	0x8BFF	384	reserviert
16-Bit-Bereich — reserviert			
0x8C00	0x8FFF	512	reserviert
8-Bit-RAM-Bereich – AVR kann schreiben und lesen			
0x9000	0x97FF	2048	RAM: 2 kByte
16-Bit-Bereich — reserviert			
0x9800	0x9BFF	512	reserviert
8-Bit-Register – AVR kann nur lesen			
	0x9C00	1	IrqFlagServo70
	0x9C01	1	IrqFlagServo158

Adr(AVR)	Ende	Words	Beschreibung
	0x9C02	1	IrqFlagServo2316
	0x9C03	1	IrqFlagsOthers
0x9C04	0x9FFF	1020	reserviert

8 Inverse Kinematik

Dieses Kapitel gibt einen kurzen Überblick über die Berechnung der inversen Kinematik des Hexapods. Die Kinematik (gr.: kinema, Bewegung) ist ein Teilgebiet der Mechanik. Sie beschäftigt sich mit der Berechnung von Punkten und Körpern im Raum.

Um die inverse Kinematik mit dem Hexapod zu verwenden, muß dieser zuvor kalibriert werden. Programmbeispiele be nden sich am Ende des Kapitels.

8.1 Allgemeines

Nachfolgend wird kurz auf die Funktion der inversen Kinematik eingegangen. Detaillierte Grundlagen der inversen Kinematik nden sich in Lehrbüchern zur (Industrie-)Robotik, wie z.B. SPONG, HUTCHINSON und VIDYASAGAR (2005); CRAIG (2004)

8.1.1 Vorwärtskinematik

Bei der Vorwärtskinematik sind die Winkel der einzelnen Gelenke bekannt, jedoch nicht der Endpunkt (Abbildung 8.1). Am Beispiel des Hexapod bedeutet das, dass die Winkel aller Servomotoren bereits gegeben sind und die Position des Fußpunktes berechnet wird.
Jedes Segment des Beins wird durch einen Vektor dargestellt. Diese Vektoren werden nun aneinander gehängt, Femur an Coxa und Tibia an Femur. Der Winkel zwischen den Vektoren ist gleich den Winkeln der Servomotoren. Hat man alle Winkel und Längen der Vektoren kann der Fußpunkt berechnet werden.

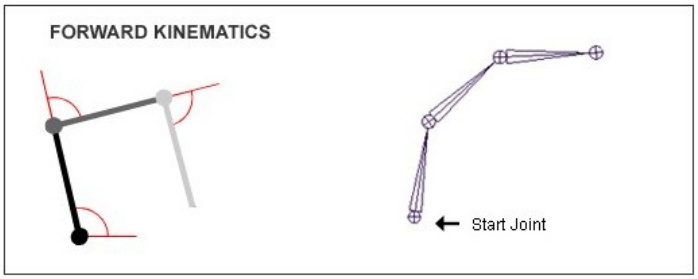


Abbildung 8.1: Vorwärtskinematik, alle Winkel sind bekannt, der Endpunkt wird berechnet.

8.1.2 Inverse Kinematik

Mit der inversen Kinematik lassen sich die Winkel bei gegebenem Endpunkt berechnen (Abbildung 8.2). Es wird also vom Endpunkt zurück zum Ursprung gerechnet. In entgegengesetzter Richtung zur Vorwärtskinematik.

Am Beispiel des Hexapod bedeutet das, dass der Fußpunkt bekannt ist, jedoch nicht die Servowinkel. Die Berechnung ist etwas komplexer wie die der Vorwärtskinematik. Da oft mehrere Winkelkombinationen zu dem selben Punkt führen.

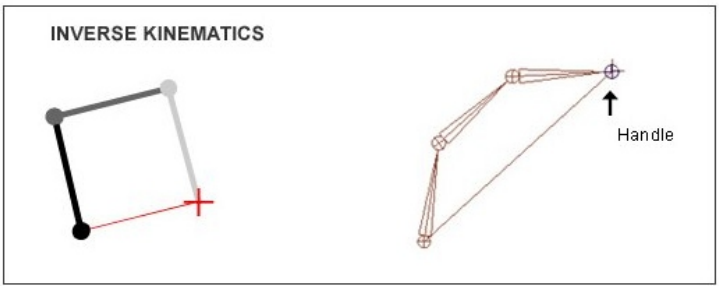


Abbildung 8.2: Inverse Kinematik, der Endpunkt ist bekannt, die Winkel werden berechnet.

8.2 Inverse Kinematik des Hexapods

In dieser Implementierung für die Mechanik des Hexapods „Captain Ahab“ werden die Winkel durch folgende Gleichungen bestimmt:

$$\varphi_0 = \arctan\left(\frac{y}{x}\right) \tag{8.1}$$

$$x_p = x \cdot \cos(\varphi_0) + y \cdot \sin(\varphi_0) - a_3 \tag{8.2}$$
$$y_p = y \cdot \cos(\varphi_0) - x \cdot \sin(\varphi_0) \tag{8.3}$$

$$D = \frac{x_p^2 + z^2 - a_1^2 - a_2^2}{2a_1a_2} \tag{8.4}$$
$$D = \frac{x_p^2 + z^2 - c_1}{c_2} \tag{8.5}$$

$$\varphi_2 = \arctan\left(\frac{\sqrt{1 - D^2}}{D}\right) \tag{8.6}$$

$$\varphi_1 = \arctan\left(\frac{z}{x_p}\right) - \arctan\left(\frac{a_2 \cdot \sin(\varphi_2)}{a_1 + a_2 \cdot \cos(\varphi_2)}\right) \tag{8.7}$$

In diesem Kapitel werden anstatt der deutschen Begriffe für die einzelnen Elemente eines Beins die in der Robotik und Insektenkunde gebräuchlicheren, lateinischen Begriffe verwendet:

- Coxa: Hüfte bzw. oberster Teil des Beines. Coxa stellt den Bereich dar, der am nächsten am Körper liegt.
- Femur: Oberschenkel bzw. Mittelstück des Beins bei der gegebenen Mechanik.
- Tibia: Unterschenkel inklusive Fußpunkt beim Hexapod.

Die Auflösung des 3-dimensionalen Koordinatensystem für den Fußpunkt beträgt etwa 1 mm. Der Ursprung dieses Koordinatensystems (Coxa-System) liegt in der Achse des Coxa-Servos und einer gedachten horizontalen Ebene durch die Achse des Femur-Servos. Das Koordinatensystem ist so gedreht, dass die positive Z-Achse des Koordinatensystems nach unten, die positive X-Achse vom Körper weg und die positive Y-Achse — bei einem von oben gesehenen Hexapod — in Richtung des Uhrzeigersinns zeigt (siehe Abb. **??**). ☐ d
Für die einfachere Programmierung eines Bewegungsablaufs werden die Koordinatensysteme aller Beine so gedreht, dass alle x- bzw. y-Achsen parallel zu einander sind (siehe Abb. **??**). Diese gedrehten Koordinatensysteme werden in der Folge als rotierte Coxa-Systeme bezeichnet. ☐ e

8.3 Hexapod Kalibrierung

Neben der Nichtlinearität der Servos — die mit Hilfe der zuvor ermittelten Werte vom RoboticsChip kompensiert werden kann — gibt es noch einen zweiten Parameter der korrigiert werden muss um sicherzustellen, dass sich alle 6 Beine des Hexapods gleich verhalten.

Das Problem ist hier allerdings nicht das Servo selbst, sondern die Scheibe welche an Servo und Spinnenbein montiert wird. Durch die zum Teil unterschiedlichen Positionen der Zähne im Innenzahnkranz der Servoscheibe und deren Befestigungslöcher, an die das Spinnenbein montiert wird, entsteht ein Differenzwinkel. Dieser Fehler kann durch versetztes Aufsetzen der Servoscheibe reduziert werden, diese Reduktion ist allerdings für Bewegungsabläufe nicht genau genug, durch die Verzahnung ergibt sich ein maximaler Differenzwinkel von ca. 7°. Diese Differenzwinkel werden mit Hilfe der Mechanik des Hexapods bestimmt. Sie müssen im Programm abgespeichert werden. Dadurch kann eine Kompensierung während der Programmlaufzeit erfolgen.
Die Nichtlinearität der Servowinkel muss jedoch bereits berücksichtigt werden. Dazu werden die im Kapitels 2 ermittelten Werte in die Datenstruktur `gTable` eingetragen. Die Datenstruktur besteht aus 24 Datensätzen (je Zeile einer), für jedes Servo einen Datensatz. Jeder Datensatz besteht aus 17 Stützstellen zur Linearisierung, einem Offset-Winkel (der nun ermittelt wird) und der Standard-Drehrichtung. Der erste Datensatz wird dem Servoanschluß SERVO0 zugeordnet, der zweite dem SERVO1, bis SERVO23.

8.3.1 Spezifikation

Mittels dem Programm *CalibrateHexapod* werden die einzelnen Beinelemente in einen Randbereich des Arbeitsraums bewegt, dessen idealer Winkel bekannt ist, dieser Winkel wird dann vom tatsächlichen Winkel subtrahiert und es ergibt sich der Differenzwinkel *PhiOffset*.
Nach dem Einschalten wird MSG_LED_0 eingeschaltet und es kann über die USB-Schnittstelle der Sandbox eine Verbindung mit dem PC hergestellt werden. Bei einem Tastendruck von AVR_KEY_0 wird eine Willkommensnachricht, sowie eine kurze Anleitung für das Programm über USB ausgegeben. So kann der Winkel über AVR_KEY_0 erhöht werden, über AVR_KEY_1 vermindert werden

bzw. durch das Drücken beider Tasten zum nächsten Servo gewechselt werden. Welches Servo gerade aktiv ist wird über eine leuchtende LED neben dem Servostecker angezeigt. Wird eine der Tasten länger gedrückt, so erhöht sich die Geschwindigkeit des Servos. Der aktuelle Winkelwert wird nach jedem Tastendruck ausgegeben.

8.3.2 Messung

Zur Messung der einzelnen Werte müssen folgende Servopositionen angefahren und die Winkelwerte aufgezeichnet werden:

- Coxa (Hüftservo): Um die Mittelposition genau bestimmten zu können, werden alle Beine zunächst völlig ausgestreckt. Danach werden, mit Hilfe eines Lineals, alle Servos so eingestellt, dass sowohl die Schraube des Coxa-Servos und der Fußpunkt beider gegenüberliegender Beine auf einer Linie liegen (Abbildung 8.3).
- Femur (Oberschenkelservo): So weit nach oben drehen, bis sich ein Streifen Papier gerade nicht mehr zwischen Coxa und Femur schieben lässt (Abbildung 8.4).
- Tibia (Unterschenkelservo): So weit nach unten drehen, bis sich ein Streifen Papier gerade nicht mehr zwischen Femur und Tibia schieben lässt (Abbildung 8.4).

Die zuvor gemessenen Werte werden nun von den unten angegebenen idealen Werten abgezogen. Daraus ergeben sich die Offset-Werte die für die Servo-Tabelle benötigt werden. Hierbei muß natürlich auf die richtige Servonummer geachtet werden.

Die Offset-Werte werden wie folgt berechnet (die zuvor ermittelten Messwerte werden in diesen Gleichungen als $\varphi_{Mitte\,Coxa}$, $\varphi_{Min\,Femur}$ und $\varphi_{Min\,Tibia}$ bezeichnet):

$$\varphi_{Offset\,Coxa} = \varphi_{Mitte\,Coxa} - 1609 \tag{8.8}$$

$$\varphi_{Offset\,Femur} = \varphi_{Min\,Femur} - 3272 \tag{8.9}$$

$$\varphi_{Offset\,Tibia} = \varphi_{Min\,Tibia} \tag{8.10}$$

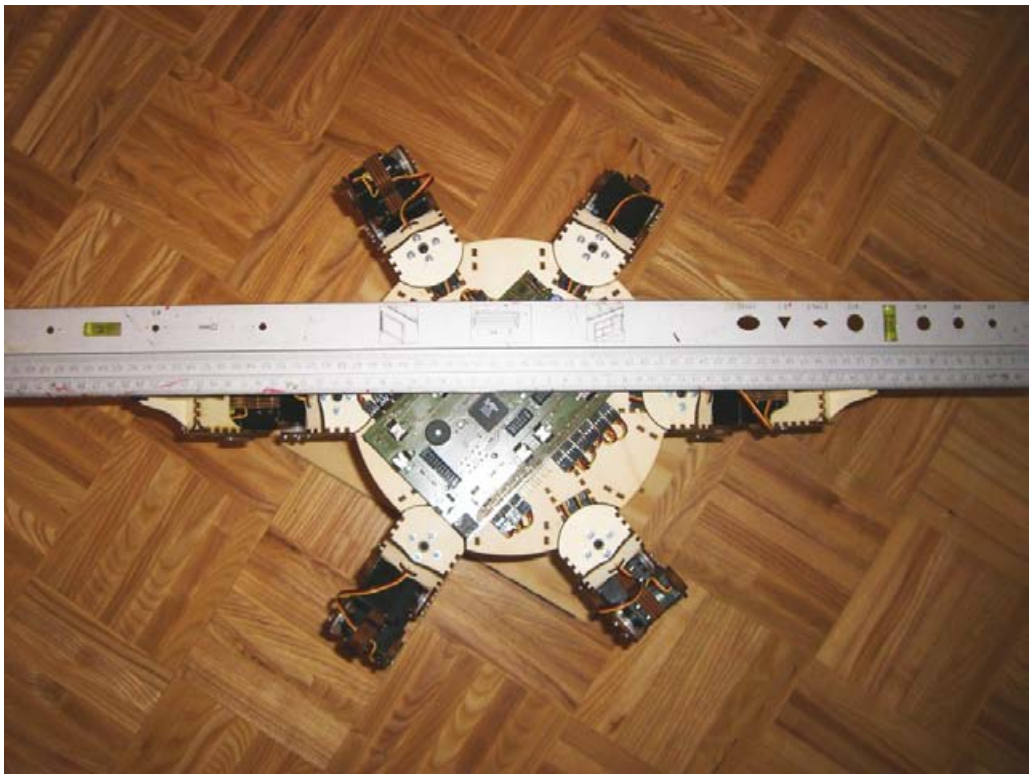


Abbildung 8.3: Kalibrierung der Coxa Servos.

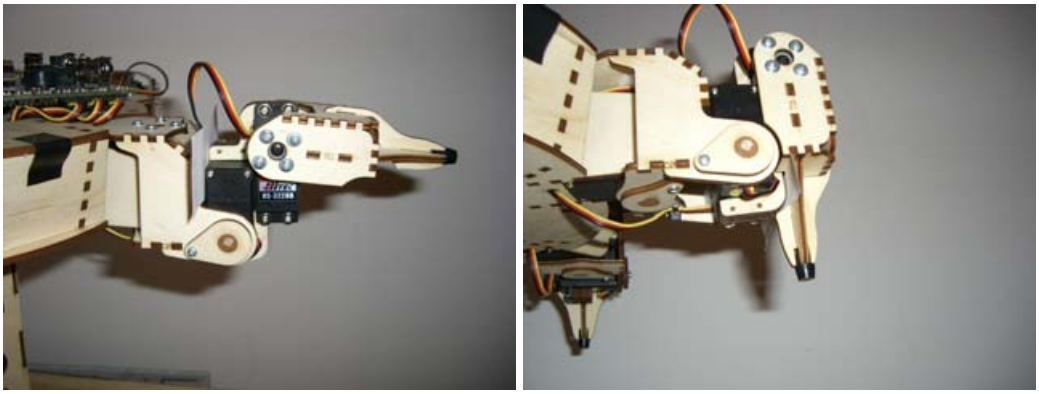


Abbildung 8.4: Kalibrierung der Femur (links) und Tibia (rechts) Servos.

8.3.3 Quellcode

```
/*****************************************************************************
*  Workfile     : main.c
*  Author         : Christian Raschko
*  Date           : 2007-08-20
*  Description: main program
*  Revision     : 3
*****************************************************************************/


//////////////////////////////////////////////////////////////////////////////
// includes
//////////////////////////////////////////////////////////////////////////////
#include <stdio.h>

#include <avr/interrupt.h>
#include <util/delay.h>
#include <avr/pgmspace.h>

#include "SandboxS2_lib.h"
#include "RoboticsChip.h"
#include "uart.h"
#include "uart_usb_lib.h"


//////////////////////////////////////////////////////////////////////////////
// global variables/definitions
//////////////////////////////////////////////////////////////////////////////
FILE * gStrmUSB  = NULL;
FILE * gStrmUART = NULL;


//Key states
typedef enum
{
  INCREMENT,
  DECREMENT,
  SELECT,
  NO_KEY
} KeysState;


//servo attribute table, remains in flash memory
RC_ServoAttributes gTable[RC_NUMBER_OF_SERVOS] PROGMEM =
{
    {{36, 385, 652, 918, 1183, 1453, 1745, 2038, 2338, 2608, 2882, 3158, 3421,
        3686, 3952, 3952, 3952},0,RC_SERVO_REVERSE_ON}, // Servo #16
    {{ 1, 277, 543, 799, 1081, 1363, 1661, 1965, 2263, 2559, 2835, 3109, 3370,
        3641, 3908, 3908, 3908},0,RC_SERVO_REVERSE_ON}, // Servo #17
    {{12, 315, 568, 822, 1100, 1378, 1652, 1941, 2255, 2555, 2837, 3101, 3362,
        3619, 3882, 3882, 3882},0,RC_SERVO_REVERSE_ON}, // Servo #18
    {{14, 255, 511, 798, 1075, 1354, 1643, 2008, 2242, 2582, 2857, 3135, 3410,
        3670, 4059, 4059, 4059},0,RC_SERVO_REVERSE_ON}, // Servo #13
```

```
    {{19, 282, 560, 793, 1082, 1364, 1653, 1942, 2242, 2543, 2837, 3096, 3359,
        3629, 3903, 3903, 3903},0,RC_SERVO_REVERSE_ON}, // Servo #14
    {{13, 292, 559, 828, 1101, 1377, 1663, 1962, 2273, 2552, 2824, 3098, 3359,
        3608, 3866, 3866, 3866},0,RC_SERVO_REVERSE_ON}, // Servo #15
    {{51, 362, 628, 887, 1158, 1425, 1708, 1996, 2306, 2598, 2875, 3160, 3425,
        3690, 3953, 3953, 3953},0,RC_SERVO_REVERSE_ON}, // default
    {{51, 362, 628, 887, 1158, 1425, 1708, 1996, 2306, 2598, 2875, 3160, 3425,
        3690, 3953, 3953, 3953},0,RC_SERVO_REVERSE_ON}, // default
    {{51, 362, 628, 887, 1158, 1425, 1708, 1996, 2306, 2598, 2875, 3160, 3425,
        3690, 3953, 3953, 3953},0,RC_SERVO_REVERSE_ON}, // default
    {{51, 361, 630, 887, 1159, 1426, 1708, 1998, 2305, 2597, 2876, 3158, 3423,
        3689, 3961, 3961, 3961},0,RC_SERVO_REVERSE_ON}, // Servo #10
    {{16, 294, 564, 844, 1125, 1406, 1690, 2000, 2307, 2611, 2891, 3181, 3455,
        3727, 3994, 3994, 3994},0,RC_SERVO_REVERSE_ON}, // Servo #11
    {{ 1, 311, 544, 824, 1094, 1362, 1657, 1967, 2291, 2606, 2890, 3162, 3426,
        3690, 3957, 3957, 3957},0,RC_SERVO_REVERSE_ON}, // Servo #12
    {{54, 399, 660, 926, 1208, 1449, 1740, 2026, 2331, 2607, 2884, 3154, 3423,
        3685, 3951, 3951, 3951},0,RC_SERVO_REVERSE_ON}, // Servo #07
    {{ 0, 278, 562, 843, 1131, 1411, 1722, 2007, 2313, 2599, 2889, 3166, 3443,
        3709, 3973, 3973, 3973},0,RC_SERVO_REVERSE_ON}, // Servo #08
    {{40, 387, 643, 922, 1193, 1462, 1749, 2033, 2336, 2605, 2877, 3138, 3398,
        3651, 3921, 3921, 3921},0,RC_SERVO_REVERSE_ON}, // Servo #09
    {{51, 362, 628, 887, 1158, 1425, 1708, 1996, 2306, 2598, 2875, 3160, 3425,
        3690, 3953, 3953, 3953},0,RC_SERVO_REVERSE_ON}, // default
    {{51, 362, 628, 887, 1158, 1425, 1708, 1996, 2306, 2598, 2875, 3160, 3425,
        3690, 3953, 3953, 3953},0,RC_SERVO_REVERSE_ON}, // default
    {{51, 362, 628, 887, 1158, 1425, 1708, 1996, 2306, 2598, 2875, 3160, 3425,
        3690, 3953, 3953, 3953},0,RC_SERVO_REVERSE_ON}, // default
    {{19, 279, 545, 822, 1102, 1399, 1694, 2005, 2340, 2573, 2867, 3140, 3413,
        3680, 3931, 3931, 3931},0,RC_SERVO_REVERSE_ON}, // Servo #04
    {{10, 299, 569, 843, 1122, 1403, 1698, 2006, 2310, 2600, 2878, 3154, 3416,
        3673, 3948, 3948, 3948},0,RC_SERVO_REVERSE_ON}, // Servo #05
    {{11, 314, 579, 842, 1134, 1419, 1699, 1994, 2279, 2545, 2818, 3089, 3359,
        3633, 3903, 3903, 3903},0,RC_SERVO_REVERSE_ON}, // Servo #06
    {{62, 382, 661, 944, 1225, 1508, 1787, 2082, 2384, 2680, 2924, 3181, 3424,
        3662, 3898, 3898, 3898},0,RC_SERVO_REVERSE_ON}, // Servo #01
    {{76, 396, 657, 901, 1184, 1452, 1749, 2033, 2332, 2625, 2892, 3158, 3410,
        3672, 3944, 3944, 3944},0,RC_SERVO_REVERSE_ON}, // Servo #02
    {{ 1, 269, 533, 797, 1065, 1344, 1641, 1954, 2256, 2548, 2844, 3130, 3409,
        3673, 3946, 3946, 3946},0,RC_SERVO_REVERSE_ON}  // Servo #03
};


////////////////////////////////////////////////////////////////////////
// functions
////////////////////////////////////////////////////////////////////////


//stream I/O wrapper functions
int strm_uart_putc(char c,FILE *s) {uart_putc(c);return 0;}
int strm_uart_getc(FILE *s)        {return (unsigned char)uart_getc();}
int strm_usb_putc(char c,FILE *s)  {uart_usb_putchar(c);return 0;}
int strm_usb_getc(FILE *s)         {return (unsigned char)uart_usb_getchar();}

//do all the initialization stuff here
void Init(void)
```

```
{
  //initialize sandboxs2 AVR
  SL2_Init();

  //initialize robotics chip
  RC_Init();

  //initialize usb
  uart_usb_init();

  //initialize uart
  uart_init(UART_CALC_BAUDRATE(38400));

  //clear all LEDs
  RC_SetLedClearAll();

  //activate stream I/O
  gStrmUSB = fdevopen(strm_usb_putc,strm_usb_getc);
  gStrmUART = fdevopen(strm_uart_putc,strm_uart_getc);

  //enable interrupts
  sei();
}


//get keys state
KeysState GetKeysState(void)
{
  uint16_t i;

  //key is still pressed
  if(SL2_AvrGetKey(SL2_KEY_0))
  {
    _delay_loop_2(0xfff);
    return INCREMENT;
  }
  if(SL2_AvrGetKey(SL2_KEY_1))
  {
    _delay_loop_2(0xfff);
    return DECREMENT;
  }

  //wait on key
  while(!SL2_AvrGetKey(SL2_KEY_0) && !SL2_AvrGetKey(SL2_KEY_1));
  _delay_loop_2(0xffff);

  //evaluate keys
  if(SL2_AvrGetKey(SL2_KEY_0) && SL2_AvrGetKey(SL2_KEY_1))
  {
    while(SL2_AvrGetKey(SL2_KEY_0) || SL2_AvrGetKey(SL2_KEY_1));
    _delay_loop_1(0xff);
    return SELECT;
  }
  else if(SL2_AvrGetKey(SL2_KEY_0))
  {
    for(i=0;i<0xfff;i++)
```

```
    {
      _delay_loop_1(0xff);
      if(!SL2_AvrGetKey(SL2_KEY_0))
        break;
    }
    return INCREMENT;
  }
  else if(SL2_AvrGetKey(SL2_KEY_1))
  {
    for(i=0;i<0xfff;i++)
    {
      _delay_loop_1(0xff);
      if(!SL2_AvrGetKey(SL2_KEY_1))
        break;
    }
    return DECREMENT;
  }

  return NO_KEY;
}


//program entry point
int main (void)
{
  uint16_t angle = 1609;
  uint8_t servo = 0;

  //Initialize system
  Init();

  // copy the linearisation table into the RoboticsChip
  RC_SetGlobalServoAttributesTable_P(gTable);

  //wait on key
  SL2_AvrSetMsgLed (SL2_MSG_LED_0, SL2_MSG_LED_STATUS_ON);
  SL2_AvrWaitOnKey(SL2_KEY_0, SL2_KEY_PRESSED);
  SL2_AvrWaitOnKey(SL2_KEY_0, SL2_KEY_NOT_PRESSED);

  //print welcome message
  fprintf(gStrmUSB,"Hexapod␣calibration␣program:\n");
  fprintf(gStrmUSB,"Press␣AvrKey0␣for␣angle␣increment,␣hold␣key␣to␣increase␣speed
      .\n");
  fprintf(gStrmUSB,"Press␣AvrKey1␣for␣angle␣decrement,␣hold␣key␣to␣increase␣speed
      .\n");
  fprintf(gStrmUSB,"Press␣AvrKey0␣and␣AvrKey1␣together␣to␣select␣next␣servo.\n");

  //set servo mode
  RC_SetGlobalServoAttributesTable_P(gTable);
  RC_SetGlobalServoMode(RC_SERVO_MODE_NO_ACTION);
  RC_SetServoMode(servo,RC_SERVO_MODE_PHI_DIRECT);
  RC_SetServoPhiDirect(servo,angle);

  //show active servo
  RC_SetLedClearAll();
  RC_SetLedBrightness(servo,0xff);
```

```
  for(;;)
  {
    switch(GetKeysState())
    {
      case INCREMENT:
        (angle == ((1 << 12) - 1) ? : angle++);
        fprintf(gStrmUSB,"Angle:␣%u\n",angle);
        break;

      case DECREMENT:
        (angle == 0 ? : angle--);
        fprintf(gStrmUSB,"Angle:␣%u\n",angle);
        break;

      case SELECT:
        servo++;
        servo %= RC_NUMBER_OF_SERVOS;
        fprintf(gStrmUSB,"Active␣servo:␣%u\n",servo);

        RC_SetGlobalServoMode(RC_SERVO_MODE_NO_ACTION);
        RC_SetServoMode(servo,RC_SERVO_MODE_PHI_DIRECT);
        RC_SetLedClearAll();
        RC_SetLedBrightness(servo,0xff);
        angle = 1609;
        break;

      default:
        break;
    }
    RC_SetServoPhiDirect(servo,angle);
  }

  //never return, loop forever
  for(;;);
    return 0;
}
```

8.4 Inverse Kematik Test

Da nun alle nötigen Servoparameter ermittelt wurden, kann das inverse Kinematik Modul getestet werden.

8.4.1 Spezifikation

Nach dem Start des Programms bewegt sich der Fußpunkt des Beins 3 auf den Startpunkt $(140, 40, 0)$. Durch Drücken der Taste Key_Avr0 wandert der Fußpunkt entlang einer Geraden (idealisiert) bis zum Endpunkt $(140, -40, 0)$. Wurde der Endpunkt erreicht und die Taste los gelassen, bewegt sich der Fußpunkt wieder geradlinig zum Startpunkt zurück. Die Werte der Datenstruktur `gTable` müssen durch die zuletzt ermittelten Werte des Hexapods ersetzt werden (Linearisierungs-Stützstellen, Offset-Winkel, Standard-Drehrichtung).

8.4.2 Quellcode

```
/*************************************************************************
*  Workfile    : main.c
*  Author      : Michael Kreuzgruber
*  Date        : 2007-08-29
*  Description : main program
*  Revision    : 3
*************************************************************************/


//////////////////////////////////////////////////////////////////////////
// includes
//////////////////////////////////////////////////////////////////////////
#include <stdio.h>

#include <avr/interrupt.h>
#include <util/delay.h>

#include "SandboxS2_lib.h"
#include "RoboticsChip.h"
#include "InverseKinematics.h"
#include "uart.h"
#include "uart_usb_lib.h"

//////////////////////////////////////////////////////////////////////////
// global variables/definitions
//////////////////////////////////////////////////////////////////////////
FILE * gStrmUSB  = NULL;
FILE * gStrmUART = NULL;


//servo attribute table, remains in flash memory
RC_ServoAttributes gTable[RC_NUMBER_OF_SERVOS] PROGMEM =
{
    {{36, 385, 652, 918, 1183, 1453, 1745, 2038, 2338, 2608, 2882, 3158, 3421,
        3686, 3952, 3952, 3952}, 171,RC_SERVO_REVERSE_ON}, // Servo #16
    {{ 1, 277, 543, 799, 1081, 1363, 1661, 1965, 2263, 2559, 2835, 3109, 3370,
        3641, 3908, 3908, 3908}, -66,RC_SERVO_REVERSE_ON}, // Servo #17
    {{12, 315, 568, 822, 1100, 1378, 1652, 1941, 2255, 2555, 2837, 3101, 3362,
        3619, 3882, 3882, 3882},   7,RC_SERVO_REVERSE_ON}, // Servo #18
    {{14, 255, 511, 798, 1075, 1354, 1643, 2008, 2242, 2582, 2857, 3135, 3410,
        3670, 4059, 4059, 4059},-157,RC_SERVO_REVERSE_ON}, // Servo #13
    {{19, 282, 560, 793, 1082, 1364, 1653, 1942, 2242, 2543, 2837, 3096, 3359,
        3629, 3903, 3903, 3903}, -99,RC_SERVO_REVERSE_ON}, // Servo #14
    {{13, 292, 559, 828, 1101, 1377, 1663, 1962, 2273, 2552, 2824, 3098, 3359,
        3608, 3866, 3866, 3866},  44,RC_SERVO_REVERSE_ON}, // Servo #15
    {{51, 362, 628, 887, 1158, 1425, 1708, 1996, 2306, 2598, 2875, 3160, 3425,
        3690, 3953, 3953, 3953},   0,RC_SERVO_REVERSE_ON}, // default
    {{51, 362, 628, 887, 1158, 1425, 1708, 1996, 2306, 2598, 2875, 3160, 3425,
        3690, 3953, 3953, 3953},   0,RC_SERVO_REVERSE_ON}, // default
    {{51, 362, 628, 887, 1158, 1425, 1708, 1996, 2306, 2598, 2875, 3160, 3425,
        3690, 3953, 3953, 3953},   0,RC_SERVO_REVERSE_ON}, // default
    {{51, 361, 630, 887, 1159, 1426, 1708, 1998, 2305, 2597, 2876, 3158, 3423,
        3689, 3961, 3961, 3961},  21,RC_SERVO_REVERSE_ON}, // Servo #10
```

```
    {{16, 294, 564, 844, 1125, 1406, 1690, 2000, 2307, 2611, 2891, 3181, 3455,
        3727, 3994, 3994, 3994}, -70,RC_SERVO_REVERSE_ON}, // Servo #11
    {{ 1, 311, 544, 824, 1094, 1362, 1657, 1967, 2291, 2606, 2890, 3162, 3426,
        3690, 3957, 3957, 3957},  74,RC_SERVO_REVERSE_ON}, // Servo #12
    {{54, 399, 660, 926, 1208, 1449, 1740, 2026, 2331, 2607, 2884, 3154, 3423,
        3685, 3951, 3951, 3951},  74,RC_SERVO_REVERSE_ON}, // Servo #07
    {{ 0, 278, 562, 843, 1131, 1411, 1722, 2007, 2313, 2599, 2889, 3166, 3443,
        3709, 3973, 3973, 3973}, -53,RC_SERVO_REVERSE_ON}, // Servo #08
    {{40, 387, 643, 922, 1193, 1462, 1749, 2033, 2336, 2605, 2877, 3138, 3398,
        3651, 3921, 3921, 3921}, 265,RC_SERVO_REVERSE_ON}, // Servo #09
    {{51, 362, 628, 887, 1158, 1425, 1708, 1996, 2306, 2598, 2875, 3160, 3425,
        3690, 3953, 3953, 3953},   0,RC_SERVO_REVERSE_ON}, // default
    {{51, 362, 628, 887, 1158, 1425, 1708, 1996, 2306, 2598, 2875, 3160, 3425,
        3690, 3953, 3953, 3953},   0,RC_SERVO_REVERSE_ON}, // default
    {{51, 362, 628, 887, 1158, 1425, 1708, 1996, 2306, 2598, 2875, 3160, 3425,
        3690, 3953, 3953, 3953},   0,RC_SERVO_REVERSE_ON}, // default
    {{19, 279, 545, 822, 1102, 1399, 1694, 2005, 2340, 2573, 2867, 3140, 3413,
        3680, 3931, 3931, 3931}, 115,RC_SERVO_REVERSE_ON}, // Servo #04
    {{10, 299, 569, 843, 1122, 1403, 1698, 2006, 2310, 2600, 2878, 3154, 3416,
        3673, 3948, 3948, 3948},  56,RC_SERVO_REVERSE_ON}, // Servo #05
    {{11, 314, 579, 842, 1134, 1419, 1699, 1994, 2279, 2545, 2818, 3089, 3359,
        3633, 3903, 3903, 3903}, -76,RC_SERVO_REVERSE_ON}, // Servo #06
    {{62, 382, 661, 944, 1225, 1508, 1787, 2082, 2384, 2680, 2924, 3181, 3424,
        3662, 3898, 3898, 3898}, 107,RC_SERVO_REVERSE_ON}, // Servo #01
    {{76, 396, 657, 901, 1184, 1452, 1749, 2033, 2332, 2625, 2892, 3158, 3410,
        3672, 3944, 3944, 3944}, -18,RC_SERVO_REVERSE_ON}, // Servo #02
    {{ 1, 269, 533, 797, 1065, 1344, 1641, 1954, 2256, 2548, 2844, 3130, 3409,
        3673, 3946, 3946, 3946}, 146,RC_SERVO_REVERSE_ON}  // Servo #03
};


////////////////////////////////////////////////////////////////////////////
// functions
////////////////////////////////////////////////////////////////////////////


//stream I/O wrapper functions
int strm_uart_putc(char c,FILE *s) {uart_putc(c);return 0;}
int strm_uart_getc(FILE *s)        {return (unsigned char)uart_getc();}
int strm_usb_putc(char c,FILE *s)  {uart_usb_putchar(c);return 0;}
int strm_usb_getc(FILE *s)         {return (unsigned char)uart_usb_getchar();}


//do all the initialization stuff here
void Init(void)
{
  //initialize sandboxs2
  SL2_Init();

  //initialize robotics chip
  RC_Init();

  //initialize inverse kinematics
  IK_Init();

  //initialize usb
```

```
  uart_usb_init();

  //initialize uart
  uart_init(UART_CALC_BAUDRATE(38400));

  //clear all LEDs
  RC_SetLedClearAll();

  //activate stream I/O
  gStrmUSB = fdevopen(strm_usb_putc,strm_usb_getc);
  gStrmUART = fdevopen(strm_uart_putc,strm_uart_getc);

  //enable interrupts
  sei();
}

//program entry point
int main (void)
{
  int8_t i = 0;

  //Initialize system
  Init();

  // copy the servo attributes table into the RoboticsChip
  RC_SetGlobalServoAttributesTable_P(gTable);

  //init position
  IK_SetLegMovementCoxa (3, 140, -40, 0, IK_KNEE_UP);

  //never return, loop forever
  for(;;)
  {
    // move leg 3 to (140, 40, 0)
    SL2_AvrWaitOnKey(SL2_KEY_0, SL2_KEY_PRESSED);
    for(i=-40; i<=40; i++)
    {
      IK_SetLegMovementCoxa (3, 140, i, 0, IK_KNEE_UP);
      _delay_ms(20);
    }

    // move leg 3 to (140, -40, 0)
    SL2_AvrWaitOnKey(SL2_KEY_0, SL2_KEY_NOT_PRESSED);
    for(i=40; i>=-40; i--)
    {
      IK_SetLegMovementCoxa (3, 140, i, 0, IK_KNEE_UP);
      _delay_ms(20);
    }
  }

  return 0;
}
```

9 Software-Funktionsbibliothek

In diesem Kapitel wird die Software-Bibliothek der SandboxS 2 beschrieben, mit welcher es möglich ist, alle Hardware-Einheiten der Platine mit geringem zusätzlichen Aufwand zu verwenden.

9.1 Übersicht der Schnittstellen

Die Software-Funktionsbibliothek umfasst folgende Funktionen:

- **Tasten und Leuchtdioden des AVR**
  Ermöglicht einfachen Zugriff auf die Tasten und Leuchtdioden des AVR.
- **Auslesen und Beschreiben zusätzlicher Pins des AVR**
  Über diese Funktionen können Daten an eine zusätzliche Hardware übermittelt werden.
- **Strom- und Spannungsmessung**
  Bietet die Möglichkeit, den aktuellen Stromverbrauch zu messen bzw. zu beurteilen, ob die Akkus neu geladen werden sollten.
- **PC-Schnittstellen (USB und UART)**
  Über diese Schnittstellen ist es möglich, Daten mit einem PC auszutauschen.
- **RoboticsChip**
  Die Funktionen dieser Gruppe ermöglichen Zugriff auf die Servos, die Servo-LEDs, die PS/2-Schnittstelle, das Sony-Playstation2-Gamepad, den Summer, die Fernsteuerung und den CORDIC-Prozessor.
- **Inverse Kinematik**
  Berechnung der inversen Kinematik eines Spinnenbeins.
- **SD-Karte**
  Mit diesen Funktionen können Daten von der SD-Karte gelesen und geschrieben werden, z.B. um Positionsdaten für Servos einfach zu verwalten.

Die einzelnen Schnittstellen und das jeweilige Verhalten werden im Folgenden erläutert.

Vorab eine Übersicht über die *Methoden bzw. Funktionen* der einzelnen *Schnittstellen*:

Modul:	**SandboxS2_lib**
Funktionalität:	*Initialisierung*
Methoden:	SL2_Init
Funktionalität:	*Tasten und Leuchtdioden des AVR*
Methoden:	SL2_AvrSetMsgLed SL2_AvrGetKey SL2_AvrWaitOnKey
Funktionalität:	*Lesen und Schreiben zusätzlicher Pins*
Methoden:	SL2_ExtAvrSetPinMode SL2_ExtAvrGetPinValue SL2_ExtAvrSetPinValue
Funktionalität:	*Strom- und Spannungsmessung*
Methoden:	SL2_GetCurrent SL2_GetCurrentLowAmpli cation SL2_GetCurrentHighAmpli cation SL2_GetBatteryVoltage SL2_ExtAvrGetAdcValue
Modul:	**uart**
Funktionalität:	*UART Kommunikation*
Methoden:	uart_init uart_init_double_speed uart_putc uart_puts uart_getc uart_is_rx_empty uart_is_tx_full
Modul:	**uart_usb_lib**
Funktionalität:	*Virtuelle serielle Schnittstelle, verwendet USB*
Methoden:	uart_usb_init uart_usb_putchar uart_usb_getchar uart_usb_test_hit uart_usb_tx_ready uart_usb_flush
Modul:	**RoboticsChip**
Funktionalität:	*Initialisierung*
Methoden:	RC_Init

Funktionalität:	*Leuchtdioden des Robotics Chips*
Methoden:	RC_SetLedBrightness
	RC_SetLedClearAll
Funktionalität:	*Servos des Robotics Chips*
	RC_SetServoMode
	RC_SetGlobalServoMode
	RC_SetServoRotation
	RC_SetServoAttributesTable
	RC_SetServoAttributesTable_P
	RC_SetServoAttributesTable_E
	RC_SetGlobalServoAttributesTable
	RC_SetGlobalServoAttributesTable_P
	RC_SetGlobalServoAttributesTable_E
	RC_SetServoPwmDirect
	RC_GetServoCurrentPwm
	RC_SetServoPhiSpeed
	RC_SetServoPhiDirect
	RC_GetServoDeltasToGo
	RC_GetServoCurrentDeltaPhi
	RC_GetServoCurrentPhi
	RC_SetServoTuneFine
	RC_SetServoTuneCoarse
Funktionalität:	*Eingabegeräte (Sony-Playstation2-Gamepad, . . .)*
	RC_GetPs2Data
	RC_SetRxControl
	RC_GetRxControl
	RC_GetRxData
	RC_GetGamePadButtons
	RC_GetGamePadJoystickData
	RC_GetGamePadData
Funktionalität:	*Buzzer (Summer)*
	RC_SetBuzzerVolume
	RC_SetBuzzerPhaseHigh
	RC_SetBuzzerPhaseLow
Funktionalität:	*CORDIC (Winkelprozessor)*
	RC_SetCordicCalculationArguments
	RC_SetCordicCalculationStart
	RC_CordicWaitOnResult
	RC_GetCordicCalculationResult
Funktionalität:	*Hilfsfunktionen*
	RC_GetRoboticsChipVersion
	RC_GetGlobalTimer
	RC_SetInterrupt0

Modul:	**InverseKinematics**
Funktionalität:	*Initialisierung*
Methoden:	IK_Init
Funktionalität:	*Berechnung der inversen Kinematik eines Spinnenbeins*
Methoden:	IK_CalculateInverseKinematicsCoxa IK_SetLegMovementCoxa IK_SetLegMovement IK_GivensTransformation

Modul:	**MemoryCard**
Funktionalität:	*Lesen und schreiben von SD-Karten*
Methoden:	mmc_init fat_cluster_data_store GetClusterSize GetBytesPerSec fat_search_ le fat_read_ le fat_write_ le

In den nachfolgenden Abschnitten werden die einzelnen Funktionen erläutert. Meist wird die Deklaration angegeben, wobei `[in]` einen Eingangsparameter, `[out]` einen Ausgangsparameter und `[inout]` einen Übergangsparameter darstellt. Weiters werden die Parameter und Rückgabewerte beschrieben und ein kurzes Beispiel gezeigt.

9.2 Datentypen

In den folgenden Funktionen werden die vorzeichenlosen Datentypen `uint8_t`, `uint16_t` und `uint32_t` sowie die vorzeichenbehafteten Datentypen `int8_t`, `int16_t` und `int32_t` verwendet. Die Zahlen `8`, `16` und `32` bestimmen den von der Variable benötigten Speicherplatz in Bit sowie deren Wertebereich. Sie beeinflussen die Anzahl der Prozessorzyklen, die eine Operation benötigt. Diese Datentypen sind der Datei *stdint.h* im Installationsordner von WinAvr de niert.

9.3 Initialisierung

Die in diesem Abschnitt beschriebene Software-Funktionsbibliothek besteht aus mehreren Modulen. Bevor ein bestimmtes Modul verwendet werden kann, muss dieses initialisiert werden. Die Funktionsaufrufe aller Initialisierungsfunktionen sind wiederum in der Funktion `Init()` in der Datei *main.c* zusammengefasst.

```
//do all the initialization stuff here
void Init(void)
{
    //initialize the RoboticsChip
    RC_Init();

    //initialize sandboxS2
    SL2_Init();

    //initialize inverse kinematics
    IK_Init();

    //initialize usb
    uart_usb_init();

    //initialize uart
    uart_init(UART_CALC_BAUDRATE(38400));

    //clear all LEDs
    RC_SetLedClearAll();

    //activate stream I/O
    gStrmUSB = fdevopen(strm_usb_putc,strm_usb_getc);
    gStrmUART = fdevopen(strm_uart_putc,strm_uart_getc);

    //enable interrupts
    sei();
}
```

Die Reihenfolge der Aufrufe muss nicht beachtet werden. Eine mehrmalige Initialisierung ist jedoch zu Vermeiden. Die Initialisierungen sollten Grundsätzlich immer am Beginn des Programms erfolgen. Im weiteren Verlauf dieses Abschnitts sind alle nötigen Initialisierungsfunktionen für jede Funktion der Software-Funktionsbibliothek angegeben, in den Beispielen sind die Aufrufe dieser Funktionen durch den Aufruf der Funktion `Init()` implementiert.

9.4 SandboxS2_lib

9.4.1 SL2_Init

```
void SL2_Init()
```

Beschreibung

Initialisiert die SandboxS 2 AVR Komponenten.

Rückgabewert

Diese Funktion hat keinen Rückgabewert.

Anmerkung

- Muss vor allen anderen Funktionen des Moduls aufgerufen werden.

9.4.2 SL2_AvrSetMsgLed

```
void SL2_AvrSetMsgLed ([in] uint8_t Number,
                       [in] uint8_t Status)
```

Beschreibung

Ein- bzw. Ausschalten der beiden Message LEDs des AVR.

Parameter	Bedeutung
Number	Gibt den Index jener Leuchtdiode(n) an, die ein- bzw. ausgeschaltet werden soll(en). Kann folgende Werte annehmen: `SL2_MSG_LED_1`, `SL2_MSG_LED_2` oder `SL2_MSG_LED_ALL`
Status	De niert ob die Leuchtdiode ein- oder ausgeschaltet werden soll. Mögliche Werte: `SL2_MSG_LED_STATUS_ON`, `SL2_MSG_LED_STATUS_OFF`

Rückgabewert

Diese Funktion hat keinen Rückgabewert.

Anmerkung

- Werden ungültige Parameter übergeben, so wird der aktuelle Status der Leuchtdioden nicht verändert.
- Bevor diese Funktion verwendet werden kann, muss `SL2_Init()` aufgerufen werden.

Beispiel

Einschalten von Leuchtdiode 1, die Leuchtdiode wird nicht wieder ausgeschaltet:

```
int main ()
{
    //function that has to be executed in every appli-
    //cation to initialize the interface with the FPGA
    SL2_Init();

    //switch on AVR LED 1
    SL2_AvrSetMsgLed (SL2_MSG_LED_1, SL2_MSG_LED_STATUS_ON);

    //ensure that the program NEVER ends (as there is no
    //operating system)
    while(1)
    {
    }

    return 0;
}
```

9.4.3 SL2_AvrGetKey

`uint8_t SL2_AvrGetKey ([in] uint8_t Number)`

Beschreibung

Überprüft, ob die Taster beim AVR gedrückt sind oder nicht.

Parameter	Bedeutung
Number	Gibt den Index des Tasters an, der eingelesen werden soll. Kann folgende Werte annehmen: `SL2_KEY_0` oder `SL2_KEY_1`

Rückgabewert

Rückgabewert	Beschreibung
SL2_KEY_NOT_PRESSED	Taste ist nicht gedrückt.
SL2_KEY_PRESSED	Taste ist gedrückt.
SL2_INVALID_PARAMETER	Parameter `Number` ist nicht gültig.

Anmerkung

Bevor diese Funktion verwendet werden kann, muss `SL2_Init()` aufgerufen werden.

Beispiel

Wenn Key 1 gedrückt ist, wird die AVR LED 1 eingeschaltet, ansonsten bleibt sie dunkel:

```
int main ()
{
    uint8_t State = SL2_KEY_NOT_PRESSED;

    //function that has to be executed in every appli-
    //cation to initialize the interface with the FPGA
    SL2_Init();

    while(1)
    {
        State = SL2_AvrGetKey (SL2_KEY_1);

        if (State == SL2_KEY_PRESSED)
        {
            SL2_AvrSetMsgLed (SL2_MSG_LED_1, SL2_MSG_LED_STATUS_ON);
        }
        else
        {
            SL2_AvrSetMsgLed (SL2_MSG_LED_1, SL2_MSG_LED_STATUS_OFF);
        }
    }

    return 0;
}
```

9.4.4 SL2_AvrWaitOnKey

```
uint8_t SL2_AvrWaitOnKey ([in] uint8_t Number,
                          [in] uint8_t State)
```

Beschreibung

Blockiert so lange die Abarbeitung des Programmes, bis ein Taster gedrückt oder losgelassen wird.

Parameter	Bedeutung
Number	Gibt den Index des Tasters an, auf den gewartet werden soll. Kann folgende Werte annehmen: `SL2_KEY_0` oder `SL2_KEY_1`
State	De niert ob darauf gewartet werden soll, dass der Taster gedrückt wird (`SL2_KEY_PRESSED`) oder losgelassen wird (`SL2_KEY_NOT_PRESSED`).

Rückgabewert

Rückgabewert	Beschreibung
SL2_INVALID_PARAMETER	Mindestens einer der angegebenen Parameter ist nicht gültig.
SL2_SUCCESSFUL	Die Funktion wurde erfolgreich ausgeführt

Anmerkung

Bevor diese Funktion verwendet werden kann, muss `SL2_Init()` aufgerufen werden.

Beispiel

Es wird so lange gewartet, bis Key 1 gedrückt wird. Sobald die Taste gedrückt wurde, wird LED 1 eingeschaltet:

```
int main ()
{
    uint8_t State = SL2_KEY_NOT_PRESSED;

    //function that has to be executed in every appli-
    //cation to initialize the interface with the FPGA
    SL2_Init();

    State = SL2_AvrWaitOnKey(SL2_KEY_0, SL2_KEY_PRESSED);

    SL2_AvrSetMsgLed (SL2_MSG_LED_1, SL2_MSG_LED_STATUS_ON);

    while(1)
    {
    }

    return 0;
}
```

9.4.5 SL2_ExtAvrSetPinMode

`void SL2_ExtAvrSetPinMode([in] uint8_t Data)`

Beschreibung

Über diese Funktion kann ausgewählt werden, ob Pin D6 als Eingang oder als Ausgang verwendet werden soll. Pin D6 ist am Erweiterungsstecker des AVR angeschlossen und wird von der SandboxS 2 selbst nicht verwendet.

Parameter	Bedeutung
Data	De niert ob Pin D6 als Eingang oder als Ausgang verwendet werden soll. Wird der Parameter auf `SL2_EXT_PIN_OUTPUT` gesetzt, so wird der Pin als Ausgang verwendet, bei `SL2_EXT_PIN_INPUT` sowie jedem anderen Wert als Eingang.

Rückgabewert

Diese Funktion hat keinen Rückgabewert.

Anmerkung

Bevor diese Funktion verwendet werden kann, muss `SL2_Init()` aufgerufen werden.

Beispiel

Siehe Beispiele von 9.4.6 und 9.4.7 ab Seite 155.

9.4.6 SL2_ExtAvrGetPinValue

`uint8_t SL2_ExtAvrGetPinValue()`

Beschreibung

Liest den aktuellen Zustand von Pin D6 ein.

Rückgabewert

Rückgabewert	Beschreibung
SL2_EXT_PIN_OFF	An Pin D6 liegt eine logische 0 (0V) an.
SL2_EXT_PIN_ON	An Pin D6 liegt eine logische 1 (3,3V) an.

Anmerkung

- Um diese Funktion verwenden zu können, muss der Pin mittels `SL2_ExtAvrSetPinMode()` als Eingang de niert werden.
- Bevor diese Funktion verwendet werden kann, muss `SL2_Init()` aufgerufen werden.

Beispiel

Von Pin D6 lesen und den Wert an LED 1 anzeigen:

```
int main ()
{
    uint8_t Data=0;

    //function that has to be executed in every appli-
    //cation to initialize the interface with the FPGA
    SL2_Init();

    SL2_ExtAvrSetPinMode(SL2_EXT_PIN_INPUT);

    while(1)
    {
        Data = SL2_ExtAvrGetPinValue();
        SL2_AvrSetMsgLed (SL2_MSG_LED_1, Data);
    }

    return 0;
}
```

9.4.7 SL2_ExtAvrSetPinValue

`void SL2_ExtAvrSetPinValue([in] uint8_t Data)`

Beschreibung

Ermöglicht das Schreiben von Daten auf Pin D6.

Parameter	Bedeutung
Data	Wird der Parameter auf `SL2_EXT_PIN_ON` gesetzt, so wird eine logische 1 an den Ausgang gelegt (3,3V), bei `SL2_EXT_PIN_OFF` sowie jedem anderen Wert eine logische 0 (0V).

Rückgabewert

Diese Funktion hat keinen Rückgabewert.

Anmerkung

- Um diese Funktion verwenden zu können, muss der Pin mittels `SL2_ExtAvrSetPinMode()` als Ausgang de niert werden.
- Bevor diese Funktion verwendet werden kann, muss `SL2_Init()` aufgerufen werden.

Beispiel

Pin D6 zuerst auf 0V und nach einem Tastendruck auf 3,3V schalten:

```
int main ()
{
    uint8_t State=SL2_KEY_NOT_PRESSED;

    //function that has to be executed in every appli-
    //cation to initialize the interface with the FPGA
    SL2_Init();

    SL2_ExtAvrSetPinMode(SL2_EXT_PIN_OUTPUT);
    SL2_ExtAvrSetPinValue(SL2_EXT_PIN_OFF);
    State = SL2_AvrWaitOnKey(SL2_KEY_0, SL2_KEY_PRESSED);
    SL2_ExtAvrSetPinValue(SL2_EXT_PIN_ON);

    while(1)
    {
    }

    return 0;
}
```

9.4.8 SL2_GetCurrent

`uint16_t SL2_GetCurrent()`

Beschreibung

Diese Funktion misst den gesamten Stromverbrauch der SandboxS 2 und gibt diesen in *mA* zurück. Die für die Strommessung nötige Verstärkung des ADC wird automatisch ermittelt.

Rückgabewert

Rückgabewert	Beschreibung
Data	Aktuell fließender Strom in *mA*.

Anmerkung

- Bevor diese Funktion verwendet werden kann, muss `SL2_Init()` aufgerufen werden.
- Das Ergebnis ist relativ ungenau und sollte nur als Richtwert dienen.

Beispiel

Einschalten von AVR LED 1, wenn der Strom bei einer Messung größer als 1A (1A = 1000mA) ist. Die LED wird nicht wieder ausgeschaltet:

```
int main ()
{
    uint16_t Current = 0;

    //function that has to be executed in every appli-
    //cation to initialize the interface with the FPGA
    SL2_Init();

    while(1)
    {
        Current = SL2_GetCurrent();
        if (Current > 1000)
        {
            SL2_AvrSetMsgLed (SL2_MSG_LED_1, SL2_MSG_LED_STATUS_ON);
        }
    }

    return 0;
}
```

9.4.9 SL2_GetCurrentLowAmplification

`uint16_t SL2_GetCurrentLowAmplification()`

Beschreibung

Diese Funktion misst den gesamten Stromverbrauch der SandboxS 2 mit einer xen internen Verstärkung von 10 und gibt diesen in *mA* zurück. Diese Verstärkung wird an den differentiellen ADC Eingängen des AVR eingestellt.

Rückgabewert

Rückgabewert	Beschreibung
Data	Aktuell fließender Strom in *mA*.

Anmerkung

- Es wird empfohlen, statt dieser Funktion `SL2_GetCurrent()` zu verwenden, da `SL2_GetCurrent()` bei niedrigen Strömen eine höhere Verstärkung verwendet und somit genauere Ergebnisse liefert.
- Bevor diese Funktion verwendet werden kann, muss `SL2_Init()` aufgerufen werden.

Beispiel

Siehe Beispiel von 9.4.8 auf Seite 156.

9.4.10 SL2_GetCurrentHighAmplification

```
uint16_t SL2_GetCurrentHighAmplification()
```

Beschreibung

Diese Funktion misst den gesamten Stromverbrauch der SandboxS 2 (mit einer xen internen Verstärkung von 200) und gibt diesen in *mA* zurück. Diese Verstärkung wird an den differentiellen ADC Eingängen des AVR eingestellt.

Rückgabewert

Rückgabewert	Beschreibung
Data	Aktuell fließender Strom in *mA*.

Anmerkung

- Es wird empfohlen, statt dieser Funktion `SL2_GetCurrent()` zu verwenden, da `SL2_GetCurrent()` bei hohen Strömen eine niedrigere Verstärkung verwendet und daher keine Überschreitung des Wertebereichs auftreten kann.
- Bevor diese Funktion verwendet werden kann, muss `SL2_Init()` aufgerufen werden.

Beispiel

Siehe Beispiel von 9.4.8 auf Seite 156.

9.4.11 SL2_GetBatteryVoltage

```
uint16_t SL2_GetBatteryVoltage()
```

Beschreibung

Diese Funktion misst die Spannung der Akkus und gibt diese in *mV* zurück. So kann im Betrieb festgestellt werden, ob die Akkus geladen werden sollten.

Rückgabewert

Rückgabewert	Beschreibung
Data	Akkusspannung in *mV*.

Anmerkung

Bevor diese Funktion verwendet werden kann, muss `SL2_Init()` aufgerufen werden.

Beispiel

Einschalten von AVR LED 1 sobald die Akkuspannung 5,5V (5,5V = 5500mV) unterschreitet:

```
int main ()
{
    uint16_t Voltage = 0;

    //function that has to be executed in every appli-
    //cation to initialize the interface with the FPGA
    SL2_Init();


    while(1)
    {
        Voltage = SL2_GetBatteryVoltage();
        if (Voltage < 5500)
        {
            SL2_AvrSetMsgLed (SL2_MSG_LED_1, SL2_MSG_LED_STATUS_ON);
        }
    }

    return 0;
}
```

9.4.12 SL2_ExtAvrGetAdcValue

`uint16_t SL2_ExtAvrGetAdcValue([in] uint8_t Number)`

Beschreibung

Misst die Spannung an einem ADC Pin des AVR.

Parameter	Bedeutung
Number	Gibt den Index des ADC an, an dem die Spannung gemessen werden soll. Kann folgende Werte annehmen: `SL2_EXT_ADC_2` (am Erweiterungsstecker des AVR angeschlossen) oder `SL2_EXT_ADC_3` (Messung der Akkuspannung).

Rückgabewert

Rückgabewert	Beschreibung
Number	16-bit Wert des ADC, aus dem der entsprechende Spannungswert berechnet werden kann.

Anmerkung

- Die von dieser Funktion zurückgegebenen Werte sind keine Spannungen sondern lediglich Zahlenwerte, die in einem bestimmten Verhältnis zur messenden Spannung stehen. Die Formel zur Umrechnung des Zahlenwerts in die entsprechende Spannung ndet sich im Datenblatt des AVR.
- Bevor diese Funktion verwendet werden kann, muss `SL2_Init()` aufgerufen werden.

Beispiel

Einschalten von AVR LED 1 wenn die Spannung am Eingang von ADC 3 den Wert 100 unterschreitet (dieser Wert entspricht nicht 100mV!):

```
int main ()
{
    uint16_t AdcVal = 0;

    //function that has to be executed in every appli-
    //cation to initialize the interface with the FPGA
    SL2_Init();

    while(1)
    {
        AdcVal = SL2_ExtAvrGetAdcValue(EXT_ADC_3);
        if (AdcVal < 100)
        {
            SL2_AvrSetMsgLed (SL2_MSG_LED_1, SL2_MSG_LED_STATUS_ON);
        }
    }

    return 0;
}
```

9.5 Schnittstellen und Datenströme (USB und UART)

Um eine gewisse Unabhängigkeit von der Hardware zu erreichen, kann für die Ausgabe die Bibliothek stdio.h aus der „avr-libc“ verwendet werden. Dabei werden so genannte Datenströme (Streams) erstellt, die dann mit Standardfunktionen benutzt werden können. Somit macht es von der Programmierung her keinen Unterschied ob man über UART oder USB kommuniziert. Für eine genauere Beschreibung wird auf das „avr-libc“ Manual verwiesen, welches mit WinAvr installiert wird. In dem folgenden Codeabschnitt wird die Erstellung und Verwendung von Streams gezeigt.

```
//////////////////////////////////////////////////////////////////////
// includes
//////////////////////////////////////////////////////////////////////
//standard input / output functions
#include <stdio.h>

#include <avr/interrupt.h>
#include "uart.h"
#include "uart_usb_lib.h"
```

```
////////////////////////////////////////////////////////////////////////////
// global variables/definitions
////////////////////////////////////////////////////////////////////////////
//global pointers to stream objects
FILE * gStrmUSB;
FILE * gStrmUART;


////////////////////////////////////////////////////////////////////////////
// functions
////////////////////////////////////////////////////////////////////////////

//stream I/O wrapper functions
int strm_uart_putc(char c,FILE *s) {uart_putc(c);return 0;}
int strm_uart_getc(FILE *s)        {return (unsigned char)uart_getc();}
int strm_usb_putc(char c,FILE *s)  {uart_usb_putchar(c);return 0;}
int strm_usb_getc(FILE *s)         {return (unsigned char)uart_usb_getchar();}


//do all the initialization stuff here
void Init(void)
{
  //initialize usb
  uart_usb_init();

  //initialize uart
  uart_init(UART_CALC_BAUDRATE(38400));

  //create stream and attatch getc/putc functions
  gStrmUSB = fdevopen(strm_usb_putc,strm_usb_getc);
  gStrmUART = fdevopen(strm_uart_putc,strm_uart_getc);

  //enable interrupts
  sei();
}

//program entry point
int main (void)
{
  char c;
  int i;
  char str[128];

  //initialize system
  Init();

  //all functions work with gStrmUSB and gStrmUART

  //print message to both devices
  fprintf(gStrmUSB,"Hello␣World!\n");
  fprintf(gStrmUART,"Hello␣World!\n");

  //read character
  c = fgetc(gStrmUSB);

  //write character
```

```
  fputc(gStrmUSB);

  //read integer
  fscanf(gStrmUSB,"%d",&i);

  //read string
  fscanf(gStrmUSB,"%s",&str);

  //print integer as hex value and string
  fprintf(gStrmUSB,"%#x␣%s",i,str);

  //...
  //for a full list of functions see avr-libc
  //manual section "Standard IO facilities"

  //never return, loop forever
  for(;;);
  return 0;
}
```

9.6 UART

Diese Implementierung verwendet einen Pufferspeicher um Sende- und Empfangsdaten zwischenzuspeichern. Bei jedem empfangenen Zeichen wird ein Interrupt ausgelöst und dieses in den Puffer kopiert. Dort kann es dann vom Hauptprogramm abgeholt werden. Das Senden eines Zeichens erfolgt ebenfalls über einen Puffer. Dabei wird ein Interrupt ausgelöst, wenn der UART frei ist und ein Zeichen vom Puffer in das Senderegister kopiert. Es kann auch eine Hardware Flusskontrolle eingestellt werden. Standardmäßig werden zwei Pins am Erweiterungsstecker des AVR benützt, die jedoch bereits von der SandboxS2_lib verwendet werden. Daher müssen diese zuerst deaktiviert werden (siehe folgende Makros: SL2_NO_ADC, SL2_NO_EXT_PIN, UART_FLOW_USE_RTS, UART_FLOW_USE_CTS).

9.6.1 uart_init

```
void uart_init([in] uint16_t baudRate)
```

Beschreibung

Initialisiert die UART.

Parameter	Bedeutung
baudRate	Gibt den Wert für das Baudrate Register an. Dieser Werte muss mit dem Makro UART_CALC_BAUDRATE berechnet werden.

Rückgabewert

Diese Funktion hat keinen Rückgabewert.

Anmerkung

- Aufgrund des verwendeten 8MHz Systemtaktes können nur folgende gängige Baudraten fehlerfrei generiert werden (siehe AT90USB1287 Datenblatt, Kapitel 18.10): 2400, 4800, 9600, 19200, 38400.

Beispiel

Initialisierung und Verwendung des UART:

```
////////////////////////////////////////////////////////////////////
// includes
////////////////////////////////////////////////////////////////////
#include <avr/interrupt.h>
#include "uart.h"

//never blocking putc
int nb_putc(char *c)
{
  if(uart_is_tx_full())
    return -1;

  uart_putc(c);
  return 0;
}

//never blocking getc
int nb_getc(char *c)
{
  if(uart_is_rx_empty())
    return -1;

  return uart_getc(c);
}


//do all the initialization stuff here
void Init(void)
{
  //initialize uart
  uart_init(UART_CALC_BAUDRATE(38400));

  //enable interrupts
  sei();
}

//program entry point
int main (void)
{
  char c;

  //initialize system
  Init();

  //read character
  c = uart_getc();
```

```
  //print character
  uart_putc(c);

  //print string
  uart_puts("blaaah");

  //never return, loop forever
  for(;;);
  return 0;
}
```

9.6.2 uart_init_double_speed

```
void uart_init_double_speed([in] uint16_t baudRate)
```

Beschreibung

Initialisiert die UART, verdoppelt jedoch den Grundtakt des Baudrate Generators.

Parameter	Bedeutung
baudRate	Gibt den Wert für das Baudrate Register an. Dieser Werte muss mit dem Makro UART_CALC_BAUDRATE_DOUBLE_SPEED berechnet werden.

Rückgabewert

Diese Funktion hat keinen Rückgabewert.

Anmerkung

- Aufgrund des verwendeten 8MHz Systemtaktes können nur folgende gängige Baudraten fehlerfrei generiert werden (siehe AT90USB1287 Datenblatt, Kapitel 18.10): 2400, 4800, 9600, 19200, 38400.

Beispiel

siehe Abschnitt 9.6.1

9.6.3 uart_putc

```
void uart_putc([in] char data)
```

Beschreibung

Sendet ein Zeichen, fügt es in den Sendepuffer ein.

Parameter	Bedeutung
data	Zeichen das gesendet werden soll.

Rückgabewert

Diese Funktion hat keinen Rückgabewert.

Anmerkung

- Diese Funktion blockiert wenn der Puffer voll ist. Was jedoch nur Auftritt, wenn keine Interrupts aktiviert sind, oder wenn die Hardware Flusskontrolle verwendet wird und die Gegenstelle nicht bereit ist.
- Bevor diese Funktion verwendet werden kann, muss `uart_init()` aufgerufen werden.

Beispiel

siehe Abschnitt 9.6.1

9.6.4 uart_puts

```
void uart_puts([in] char* str)
```

Beschreibung

Sendet eine Zeichenfolge, fügt sie in den Sendepuffer ein.

Parameter	Bedeutung
str	Pointer einer Zeichenfolge.

Rückgabewert

Diese Funktion hat keinen Rückgabewert.

Anmerkung

- Diese Funktion blockiert wenn der Puffer voll ist. Was jedoch nur Auftritt, wenn keine Interrupts aktiviert sind, oder wenn die Hardware Flusskontrolle verwendet wird und die Gegenstelle nicht bereit ist.
- Bevor diese Funktion verwendet werden kann, muss `uart_init()` aufgerufen werden.

Beispiel

siehe Abschnitt 9.6.1

9.6.5 uart_getc

```
char uart_getc()
```

Beschreibung

Empfängt ein Zeichen, entfernt es aus dem Empfangspuffer.

Rückgabewert

Rückgabewert	Beschreibung
Data	Gibt das empfangene Zeichen zurück.

Anmerkung

- Diese Funktion blockiert wenn der Puffer leer ist. Sind keine Interrupts aktiviert können auch keine Zeichen empfangen werden. Wird die Flusskontrolle verwendet wird der Gegenstelle mitgeteilt das sie keine Daten mehr senden soll, bis der Puffer wieder leer ist.
- Bevor diese Funktion verwendet werden kann, muss `uart_init()` aufgerufen werden.

Beispiel

siehe Abschnitt 9.6.1

9.6.6 uart_is_rx_empty

```
char uart_is_rx_empty()
```

Beschreibung

Prüft ob der Empfangspuffer leer ist.

Rückgabewert

Rückgabewert	Beschreibung
Data	Gibt Null zurück wenn der Puffer nicht leer ist. Sonst eine Zahl ungleich Null.

Anmerkung

- Bevor diese Funktion verwendet werden kann, muss `uart_init()` aufgerufen werden.

Beispiel

siehe Abschnitt 9.6.1

9.6.7 uart_is_tx_full

```
char uart_is_tx_full()
```

Beschreibung

Prüft ob der Sendepuffer voll ist.

Rückgabewert

Rückgabewert	Beschreibung
Data	Gibt Null zurück wenn der Puffer nicht voll ist. Sonst eine Zahl ungleich Null.

Anmerkung

- Bevor diese Funktion verwendet werden kann, muss `uart_init()` aufgerufen werden.

Beispiel

siehe Abschnitt 9.6.1

9.7 USB-UART

Dieses Modul verwendet den integrierten USB-Kern des Mikrocontrollers um eine virtuelle serielle Schnittstelle zu simulieren. Hierfür wird die „Communication Device Class (CDC)“ des USB Standards verwendet. Um die Schnittstelle an einem Computer zu verwenden muss der Treiber vorher installiert werden. Danach kann sie, wie eine „echte“ serielle Schnittstelle, verwendet werden.

Die Übertragung erfolgt bei USB mit Datenpaketen xer Länge (von einem Endpunkt zum Host) und ist bereits mit Fehlererkennung und Flusskontrolle ausgestattet. Um die Datenblöcke empfangen und senden zu können ist ein Puffer notwendig. Dieser wird immer mit xer Größe empfangen oder gesendet, auch wenn er nur zur Hälfte voll ist.

Bei dieser Implementierung wird der Puffer jede Millisekunde (bei jedem „Start Of Frame (SOF)“ Paket) gesendet. Zusätzlich kann jedoch mit der Funktion `uart_flush()` ein manuelles Senden eingeleitet werden. Die gesamte Steuerung des Gerätes am Bus erfolgt über Interrupts. Alle Anfragen vom Host werden in Interrupt-Routinen ausgewertet und bearbeitet. Für den Anwender entsteht das Bild einer „normalen“ seriellen Schnittstelle.

9.7.1 uart_usb_init

```
void uart_usb_init()
```

Beschreibung

Initialisiert die virtuelle serielle Schnittstelle.

Rückgabewert

Diese Funktion hat keinen Rückgabewert.

Anmerkung

- Nach der Initialisierung müssen die Interrupts eingeschaltet werden.

Beispiel

Initialisierung und Verwendung des USB-UART:

```
////////////////////////////////////////////////////////////////////////
// includes
////////////////////////////////////////////////////////////////////////
#include <avr/interrupt.h>
#include "uart_usb_lib.h"

//never blocking putc
int nb_putc(char *c)
{
  if(uart_is_tx_ready())
    return -1;

  uart_usb_putchar(c);
  return 0;
}

//never blocking getc
int nb_getc(char *c)
{
  if(uart_test_hit())
    return -1;

  return uart_usb_getchar(c);
}


//do all the initialization stuff here
void Init(void)
{
  //initialize usb uart
  uart_usb_init();

  //enable interrupts
  sei();
}

//program entry point
int main (void)
{
  char c;

  //initialize system
  Init();
```

```
  //read character
  c = uart_usb_getchar();

  //print character
  uart_usb_putchar(c);

  //never return, loop forever
  for(;;);
    return 0;
}
```

9.7.2 uart_usb_putchar

`void uart_usb_putchar([in] char data)`

Beschreibung

Sendet ein Zeichen, fügt es in den Sendepuffer ein.

Parameter	Bedeutung
data	Zeichen welches gesendet werden soll.

Rückgabewert

Diese Funktion hat keinen Rückgabewert.

Anmerkung

- Diese Funktion blockiert wenn der Puffer voll ist. Jedoch nur wenn das Gerät am Bus hängt. Besteht keine Verbindung zwischen PC und Controller, werden die Daten ignoriert. Ansonst wird die Flusskontrolle schlagend und die Funktion blockiert, wenn die Daten nicht vom Host abgeholt werden.
- Bevor diese Funktion verwendet werden kann, muss `uart_usb_init()` aufgerufen werden.

Beispiel

siehe Abschnitt 9.7.1

9.7.3 uart_usb_getchar

`char uart_usb_getchar()`

Beschreibung

Empfängt ein Zeichen, entfernt es aus dem Empfangspuffer.

Rückgabewert

Rückgabewert	Beschreibung
Data	Gibt das empfangene Zeichen zurück.

Anmerkung

- Diese Funktion blockiert wenn der Puffer leer ist. Auch wenn keine Verbindung zwischen PC und Gerät besteht.
- Bevor diese Funktion verwendet werden kann, muss `uart_usb_init()` aufgerufen werden.

Beispiel

siehe Abschnitt 9.7.1

9.7.4 uart_usb_test_hit

```
char uart_usb_test_hit()
```

Beschreibung

Prüft ob der Empfangspuffer leer ist, ob Daten gelesen werden können

Rückgabewert

Rückgabewert	Beschreibung
Data	Gibt Null zurück wenn der Puffer nicht leer ist. Sonst eine Zahl ungleich Null.

Anmerkung

- Bevor diese Funktion verwendet werden kann, muss `uart_usb_init()` aufgerufen werden.

Beispiel

siehe Abschnitt 9.7.1

9.7.5 uart_usb_tx_ready

```
char uart_usb_tx_ready()
```

Beschreibung

Prüft ob der Sendepuffer voll ist, ob Daten geschrieben werden können.

Rückgabewert

Rückgabewert	Beschreibung
Data	Gibt Null zurück wenn der Puffer nicht voll ist. Sonst eine Zahl ungleich Null.

Anmerkung

- Bevor diese Funktion verwendet werden kann, muss `uart_usb_init()` aufgerufen werden.

Beispiel

siehe Abschnitt 9.7.1

9.7.6 uart_usb_flush

```
void uart_usb_flush()
```

Beschreibung

Sendet denn Puffer, gibt ihn zum Senden frei.

Rückgabewert

Diese Funktion hat keinen Rückgabewert.

Anmerkung

- Da diese Funktion jede Millisekunde aufgerufen wird, sollte man sie manuell nicht benutzen.
- Bevor diese Funktion verwendet werden kann, muss `uart_usb_init()` aufgerufen werden.

Beispiel

siehe Abschnitt 9.7.1

9.8 RoboticsChip

9.8.1 RC_Init

```
void RC_Init()
```

Beschreibung

Initialisiert die Verbindung mit dem Robotics Chip.

Rückgabewert

Diese Funktion hat keinen Rückgabewert.

Anmerkung

- Muss vor allen anderen Funktionen des Moduls aufgerufen werden.

9.8.2 RC_SetLedBrightness

```
uint8_t RC_SetLedBrightness([in] uint8_t LedNumber,
                            [in] uint8_t Brightness)
```

Beschreibung

Verändert die Helligkeit einer bestimmten Servo LED (= sich am Rand der Platine be ndende LED, neben den Steckern für die Servos).

Parameter	Bedeutung
LedNumber	Nummer der LED, deren Helligkeit verändert werden soll. Die Nummerierung der LEDs ist in Abb. 7.1 auf Seite 81 ersichtlich.
Brightness	Bestimmt die Helligkeit einer bestimmten Servo LED (0 = ausgeschaltet, 255 = maximale Helligkeit).

Rückgabewert

Diese Funktion hat keinen Rückgabewert.

Anmerkung

- Da der nichtlineare Zusammenhang zwischen der Pulslänge der PWM und der Helligkeit der Leuchtdiode nicht vom RoboticsChip kompensiert wird, ist die unterschiedliche Helligkeit nicht bei allen Stufen zu erkennen.
- Bevor diese Funktion verwendet werden kann, müssen `SL2_Init()` und `RC_Init()` aufgerufen werden.

Beispiel

Alle LEDs mit maximaler Helligkeit einschalten:

```
int main ()
{
    uint8_t i = 0;

    //proxy function for whole initialization
    Init();

    for (i = 0; i < RC_NUMBER_OF_LEDS; i++)
    {
```

```
        RC_SetLedBrightness(i, 255);
    }

    while(1)
    {
    }

    return 0;
}
```

9.8.3 RC_SetLedClearAll

```
void RC_SetLedClearAll()
```

Beschreibung

Schaltet alle Servo LEDs auf dunkel. Die LEDs des AVR werden von dieser Funktion nicht beeinflusst.

Rückgabewert

Diese Funktion hat keinen Rückgabewert.

Anmerkung

Bevor diese Funktion verwendet werden kann, müssen `SL2_Init()` und `RC_Init()` aufgerufen werden.

Beispiel

Alle Servo LEDs ausschalten:

```
int main ()
{
    //proxy function for whole initialization
    Init();

    RC_SetLedClearAll();

    while(1)
    {
    }

    return 0;
}
```

9.8.4 RC_SetServoMode

```
inline void RC_SetServoMode([in] uint8_t ServoNumber,
                            [in] uint8_t Mode)
```

Beschreibung

Schaltet ein bestimmtes Servo in den angegebenen Bewegungsmodus.

Parameter	Bedeutung
ServoNumber	Nummer des Servos, dessen Modus verändert werden soll. Die Nummerierung der Servos ist in Abb. 7.1 auf Seite 81 ersichtlich.
Mode	Bestimmt den Modus des Servos. Dieser Parameter kann folgende Werte annehmen: • `RC_SERVO_MODE_PWM_DIRECT`: (PwmDirect Modus, siehe Abschnitt 7.5 auf Seite 92 für genauere Erklärung), • `RC_SERVO_MODE_PHI_DIRECT`: (PhiDirect bzw. PhiSpeed Modi, siehe Abschnitt 7.5 auf Seite 92 für genauere Erklärung) oder • `RC_SERVO_MODE_NO_ACTION`: (Servo deakiviert)

Rückgabewert

Diese Funktion hat keinen Rückgabewert.

Anmerkung

- Der Modus `RC_SERVO_MODE_NO_ACTION` sollte nur verwendet werden, wenn der entsprechende Servo nicht belastet wird — auch nicht durch das Eigengewicht des Spinnenkörpers!
- Bevor diese Funktion verwendet werden kann, müssen `SL2_Init()` und `RC_Init()` aufgerufen werden.

Beispiel

Servo 0 in den PwmDirect Modus schalten:

```
int main ()
{
    uint8_t i = 0;

    //proxy function for whole initialization
    Init();

    //activate servo #0
    RC_SetServoMode(0, RC_SERVO_MODE_PWM_DIRECT);

    //move servo #0 to its middle position
    RC_SetServoPwmDirect (i, 2048);

    while(1)
    {
    }

    return 0;
}
```

9.8.5 RC_SetGlobalServoMode

`inline void RC_SetGlobalServoMode([in] uint8_t Mode)`

Beschreibung

Schaltet alle Servos in den angegebenen Modus.

Parameter	Bedeutung
Mode	Bestimmt den Modus des Servos. Dieser Parameter kann folgende Werte annehmen: • `RC_SERVO_MODE_PWM_DIRECT`: (PwmDirect Modus, siehe Abschnitt 7.5 auf Seite 92 für genauere Erklärung), • `RC_SERVO_MODE_PHI_DIRECT`: (PhiDirect bzw. PhiSpeed Modi, siehe Abschnitt 7.5 auf Seite 92 für genauere Erklärung) oder • `RC_SERVO_MODE_NO_ACTION`: (Servo deakiviert)

Rückgabewert

Diese Funktion hat keinen Rückgabewert.

Anmerkung

- Der Modus `RC_SERVO_MODE_NO_ACTION` sollte nur verwendet werden, wenn der entsprechende Servo nicht belastet wird — auch nicht durch das Eigengewicht des Spinnenkörpers!
- Bevor diese Funktion verwendet werden kann, müssen `SL2_Init()` und `RC_Init()` aufgerufen werden.

Beispiel

Alle Servos in den PwmDirect Modus schalten:

```
int main ()
{
    uint8_t i = 0;

    //proxy function for whole initialization
    Init();

    //activate all servos
    RC_SetGlobalServoMode(RC_SERVO_MODE_PWM_DIRECT);

    //move all servos to their middle position
    for (i=0; i<RC_NUMBER_OF_SERVOS; ++i)
    {
        RC_SetServoPwmDirect (i, 2048);
    }
```

```
    while(1)
    {
    }

    return 0;
}
```

9.8.6 RC_SetServoPwmDirect

```
inline void RC_SetServoPwmDirect([in] uint8_t ServoNumber,
                                 [in] uint16_t Length)
```

Beschreibung

Die Funktion ermöglicht die Bewegung eines Servos direkt über die Pulslänge zu steuern.

Parameter	Bedeutung
ServoNumber	Nummer des Servos, das bewegt werden soll. Die Nummerierung der Servos ist in Abb. 7.1 auf Seite 81 ersichtlich.
Length	Pulslänge; muss im Bereich 0 – 4095 liegen

Rückgabewert

Diese Funktion hat keinen Rückgabewert.

Anmerkung

- Die Drehrichtung des Servos ist nicht bei allen Herstellern bzw. Servomodellen gleich.
- Um `RC_SetServoPwmDirect()` verwenden zu können, muss der angegebene Servo zuvor in den Modus `RC_SERVO_MODE_PWM_DIRECT` geschalten werden.
- Bevor diese Funktion verwendet werden kann, müssen `SL2_Init()` und `RC_Init()` aufgerufen werden.

Beispiel

Die Posititon von Servo 0 wird bei jedem Tastendruck von AVR KEY 0 um 90° verändert:

```
int main ()
{
    //proxy function for whole initialization
    Init();

    //clear all LEDs
    RC_SetLedClearAll();

    RC_SetServoMode(0, RC_SERVO_MODE_PWM_DIRECT);

    while(1)
    {
```

```
        RC_SetServoPwmDirect(0, 0);

        SL2_AvrWaitOnKey (SL2_KEY_0, SL2_KEY_PRESSED);
        SL2_AvrWaitOnKey (SL2_KEY_0, SL2_KEY_NOT_PRESSED);

        RC_SetServoPwmDirect(0, 1609);

        SL2_AvrWaitOnKey (SL2_KEY_0, SL2_KEY_PRESSED);
        SL2_AvrWaitOnKey (SL2_KEY_0, SL2_KEY_NOT_PRESSED);
    }
    return 0;
}
```

9.8.7 RC_GetServoCurrentPwm

`uint16_t RC_GetServoCurrentPwm ([in] uint8_t ServoNumber)`

Beschreibung

Ermöglicht die mittels `RC_SetServoPwmDirect()` eingestellte Pulslänge (Length) zu lesen.

Parameter	Bedeutung
ServoNumber	Nummer des Servos, das bewegt werden soll. Die Nummerierung der Servos ist in Abb. 7.1 auf Seite 81 ersichtlich.

Rückgabewert

Rückgabewert	Beschreibung
Length	Jene Pulslänge, die über die Funktion `RC_SetServoPwmDirect()` eingestellt wurde.

Anmerkung

- Um `RC_GetServoCurrentPwm()` verwenden zu können, muss der angegebene Servo zuvor in den Modus `RC_SERVO_MODE_PWM_DIRECT` geschalten werden.
- Wenn `RC_GetServoCurrentPwm()` direkt nach `RC_SetServoPwmDirect()` aufgerufen wird, so wird nicht der soeben geschriebene Wert zurückgegeben, sondern noch der „alte“ Wert, da der RoboticsChip nur alle 20 ms die Daten aus den internen Registern „abholt“.
- Bevor diese Funktion verwendet werden kann, müssen `SL2_Init()` und `RC_Init()` aufgerufen werden.

Beispiel

Schreiben einer PWM Pulslänge, diese wieder lesen und mit dem geschriebenen Wert vergleichen:

```
int main ()
{
    uint16_t Data = 0;

    //proxy function for whole initialization
    Init();

    //clear all LEDs
    RC_SetLedClearAll();

    RC_SetServoMode(0, RC_SERVO_MODE_PWM_DIRECT);
    RC_SetServoPwmDirect(0, 1609);

    //Delay is needed to ensure that the command
    //has been executed by the RoboticsChip
    _delay_ms(100);

    Data = RC_GetServoCurrentPwm(0);

    //signal that everything is ok
    if (Data == 1609)
    {
        RC_SetLedBrightness(0, 255);
    }

    while(1)
    {
    }
    return 0;
}
```

9.8.8 RC_SetServoPhiSpeed

```
inline void RC_SetServoPhiSpeed([in] uint8_t ServoNumber,
                                [in] uint16_t InitialAngle,
                                [in] int16_t DeltaPhi,
                                [in] uint16_t NumberOfDeltas)
```

Beschreibung

Ermöglicht die Bewegung eines Servos im PhiSpeed-Modus. Der Modus ist in Abschnitt 7.7 auf Seite 102 erklärt.

Parameter	Bedeutung
ServoNumber	Nummer des Servos, das bewegt werden soll. Die Nummerierung der Servos ist in Abb. 7.1 auf Seite 81 ersichtlich.
InitialAngle	Startwinkel der Bewegung.
DeltaPhi	„Winkel“ um welchen das Servo innerhalb von 20 ms bewegt wird.
NumberOfDeltas	Anzahl der Winkelinkremente.

Rückgabewert

Diese Funktion hat keinen Rückgabewert.

Anmerkung

- Um `RC_SetServoPhiSpeed()` verwenden zu können, muss das angegebene Servo zuvor in den Modus `RC_SERVO_MODE_PHI_DIRECT` geschalten werden.
- Der Wert von DeltaPhi wird im RoboticsChip als binäre Dezimalzahl mit 4 Nachkommastellen interpretiert. Wenn eine ganze Zahl angegeben werden soll, muss diese mit 16 multipliziert werden (Schiebeoperation um 4 Bits nach links).
- Die Summe von `InitialAngle + DeltaPhi*NumberOfDeltas` muss im Bereich 0 – 4095 liegen
- Bevor diese Funktion verwendet werden kann, müssen `SL2_Init()` und `RC_Init()` aufgerufen werden.

Beispiel

Das Servo von der Position 0 aus auf die Mittelposition stellen.

```
int main ()
{
    //proxy function for whole initialization
    Init();

    //clear all LEDs
    RC_SetLedClearAll();

    //move servo 0
    RC_SetServoMode(0, RC_SERVO_MODE_PHI_DIRECT);

    RC_SetServoPhiSpeed(0, 0, 16<<4, 128);
    //time needed for movement: 128 Steps * 20 ms = 2,56 sec

    while(1)
    {
    }

    return 0;
}
```

9.8.9 RC_SetServoPhiDirect

```
inline void RC_SetServoPhiDirect([in] uint8_t ServoNumber,
                                 [in] uint16_t Angle)
```

Beschreibung

Ermöglicht die Bewegung eines Servos im PhiDirect-Modus. Der Modus ist in Abschnitt 7.7 auf Seite 102 erklärt.

Parameter	Bedeutung
ServoNumber	Nummer des Servos, das bewegt werden soll. Die Nummerierung der Servos ist in Abb. 7.1 auf Seite 81 ersichtlich.
Angle	Endwinkel der Bewegung im Bereich 0 – 4095.

Rückgabewert

Diese Funktion hat keinen Rückgabewert.

Anmerkung

- Um `RC_SetServoPhiDirect()` verwenden zu können, muss der angegebene Servo zuvor in den Modus `RC_SERVO_MODE_PHI_DIRECT` geschalten werden.
- Bevor diese Funktion verwendet werden kann, müssen `SL2_Init()` und `RC_Init()` aufgerufen werden.

Beispiel

Die Positition von Servo 0 wird bei jedem Tastendruck von AVR KEY 0 um 90° verändert:

```
int main ()
{
    //proxy function for whole initialization
    Init();

    //clear all LEDs
    RC_SetLedClearAll();

    RC_SetServoMode(0, MODE_PHI_DIRECT);


    while(1)
    {
        RC_SetServoPhiDirect(0, 0);

        SL2_AvrWaitOnKey (SL2_KEY_0, SL2_KEY_PRESSED);
        SL2_AvrWaitOnKey (SL2_KEY_0, SL2_KEY_NOT_PRESSED);

        SL2_SetServoPhiDirect(0, 1609);

        SL2_AvrWaitOnKey (SL2_KEY_0, SL2_KEY_PRESSED);
        SL2_AvrWaitOnKey (SL2_KEY_0, SL2_KEY_NOT_PRESSED);
    }

    return 0;
}
```

9.8.10 RC_GetServoDeltasToGo

`inline uint16_t RC_GetServoDeltasToGo([in] uint8_t ServoNumber)`

Beschreibung

Liest die Anzahl der noch abzuarbeitenden Winkelschritte aus dem RoboticsChip. Diese Funktion liefert nur sinnvolle Werte, wenn das angegebene Servo im Modus PhiSpeed arbeitet.

Parameter	Bedeutung
ServoNumber	Nummer des Servos, dessen Anzahl noch abzuarbeitender Winkelschritte gelesen werden soll. Die Nummerierung der Servos ist in Abb. 7.1 auf Seite 81 ersichtlich.

Rückgabewert

Rückgabewert	Beschreibung
Deltas	Anzahl der noch abzuarbeitenden Winkelinkremente.

Anmerkung

- Um `RC_GetServoDeltasToGo()` verwenden zu können, muss der angegebene Servo zuvor in den Modus `RC_SERVO_MODE_PHI_DIRECT` geschalten werden.
- Wenn `RC_GetServoDeltasToGo()` direkt nach `RC_SetServoPhiSpeed()` aufgerufen wird, so wird nicht der soeben geschriebene Wert für die Anzahl der Winkelschritte zurückgegeben, da der RoboticsChip nur alle 20 ms die Daten aus den Registern „abholt“.
- Bevor diese Funktion verwendet werden kann, müssen `SL2_Init()` und `RC_Init()` aufgerufen werden.

Beispiel

Die Werte aus dem `DeltasToGo`-Register werden an den LEDs 0 – 5 angezeigt. Dabei ist jeder LED ein bestimmter Wert für DeltasToGo zugeordnet, bei dem die Leuchtdiode eingeschaltet wird.

```
int main ()
{
    uint16_t Data = 0;
    uint8_t LedIdx = 0;

    //proxy function for whole initialization
    Init();

    //clear all LEDs
    RC_SetLedClearAll();

    RC_SetServoMode(0, RC_SERVO_MODE_PHI_DIRECT);

    RC_SetServoPhiSpeed(0, 0, 2<<4, 1024);
    //time needed for movement: 1024 Steps * 20 ms = 20,5 sec
```

```
    //wait for 20ms to ensure that the command has been
    //executed by the RoboticsChip
    _delay_ms(20);

    while(1)
    {
        //show the changes in DeltasToGo register
        //on the first 6 servo LEDs
        Data = RC_GetServoDeltasToGo(0);

        //switch on LED 0 when Data =   0
        //switch on LED 1 when Data = 200
        //switch on LED 2 when Data = 400
        //...
        LedIdx = Data / 200;
        RC_SetLedBrightness(LedIdx, 255);
    }

    return 0;
}
```

9.8.11 RC_GetServoCurrentDeltaPhi

`inline uint16_t RC_GetServoCurrentDeltaPhi([in] uint8_t ServoNumber)`

Beschreibung

Liest das Winkelinkrement eines Servos aus dem RoboticsChip.

Parameter	Bedeutung
ServoNumber	Nummer des Servos, dessen Winkelinkrement gelesen werden soll. Die Numerierung der Servos ist in Abb. 7.1 auf Seite 81 ersichtlich.

Rückgabewert

Rückgabewert	Beschreibung
DeltaPhi	Winkelinkrement des Servos.

Anmerkung

- Um `RC_GetServoCurrentDeltaPhi()` verwenden zu können, muss der angegebene Servo zuvor in den Modus `RC_SERVO_MODE_PHI_DIRECT` geschalten werden.
- Wenn `RC_GetServoCurrentDeltaPhi()` direkt nach `RC_SetServoPhiSpeed()` aufgerufen wird, so wird nicht das neue Winkelinkrement zurückgegeben, da der RoboticsChip nur alle 20 ms die Daten aus den Registern „abholt“.
- Bevor diese Funktion verwendet werden kann, müssen `SL2_Init()` und `RC_Init()` aufgerufen werden.

Beispiel

Die Übereinstimmung zwischen gelesenem und geschriebenem Wert wird durch Leuchten von LED 0 angezeigt:

```
int main ()
{
    uint16_t Data = 0;

    //proxy function for whole initialization
    Init();

    //clear all LEDs
    RC_SetLedClearAll();

    RC_SetServoMode(0, RC_SERVO_MODE_PHI_DIRECT);

    RC_SetServoPhiSpeed(0, 0, 2<<4, 1024);
    //time needed for movement: 1024 Steps * 20 ms = 20,5 sec
    _delay_ms(20);

    Data = RC_GetServoCurrentDeltaPhi(0);
    if (Data == 2<<4)
    {
        RC_SetLedBrightness(0, 255);
    }

    while(1)
    {
    }

    return 0;
}
```

9.8.12 RC_GetServoCurrentPhi

```
inline uint16_t RC_GetServoCurrentPhi([in] uint8_t ServoNumber)
```

Beschreibung

Liest die aktuelle Position aus dem RoboticsChip.

Parameter	Bedeutung
ServoNumber	Nummer des Servos, dessen Position gelesen werden soll. Die Nummerierung der Servos ist in Abb. 7.1 auf Seite 81 ersichtlich.

Rückgabewert

Rückgabewert	Beschreibung
Phi	Position des Servos (4095≙180°, 0≙0°).

Anmerkung

- Um `RC_GetServoCurrentPhi()` verwenden zu können, muss der angegebene Servo zuvor in den Modus `RC_SERVO_MODE_PHI_DIRECT` geschalten werden.
- Wenn `RC_GetServoCurrentPhi()` direkt nach `RC_SetServoPhiSpeed()` aufgerufen wird, so wird nicht der Startwinkel zurückgegeben, da der RoboticsChip nur alle 20 ms die Daten aus den Registern „abholt“.
- Bevor diese Funktion verwendet werden kann, müssen `SL2_Init()` und `RC_Init()` aufgerufen werden.

Beispiel

Die Werte aus dem `CurrentPhi`-Register werden an den ersten 11 Servo LEDs angezeigt. Dabei ist jeder LED ein bestimmter Winkel zugeordnet. Wird dieser Winkel überschritten, so wird die LED eingeschaltet. Dies hat zur Folge, dass während der Bewegung des Servos die LEDs nacheinander eingeschaltet werden:

```
int main ()
{
    uint16_t Data = 0;
    uint8_t LedIdx = 0;

    //proxy function for whole initialization
    Init();

    //clear all LEDs again
    RC_SetLedClearAll();

    RC_SetServoMode(0, MODE_PHI_DIRECT);

    RC_SetServoPhiSpeed(0, 0, 2<<4, 1024);
    //time needed for movement: 1024 Steps * 20 ms = 20,5 sec
    _delay_ms(20);

    while(1)
    {
        //maximum value: 2048 (=2*1024)
        //number of LEDs: 2048 / 200 = 10,24 -> 11 LEDs [0 - 10]
        Data = RC_GetServoCurrentPhi(0);

        //skip decimals behind the comma
        Data = Data >> 4;

        //calc the LED to switch on
        LedIdx = Data / 200;
        RC_SetLedBrightness(LedIdx, 255);
    }

    return 0;
}
```

9.8.13 RC_SetServoRotation

```
inline void RC_SetServoRotation([in] uint8_t ServoNumber,
                                [in] RC_ServoReverse Reverse)
```

Beschreibung

Schaltet die Drehrichtung eines bestimmten Servos um.

Parameter	Bedeutung
ServoNumber	Nummer des Servos, dessen Modus verändert werden soll. Die Nummerierung der Servos ist in Abb. 7.1 auf Seite 81 ersichtlich.
Reverse	Umkehr der Drehrichtung. Kann entweder auf `RC_SERVO_REVERSE_ON` oder `RC_SERVO_REVERSE_OFF` gesetzt werden.

Rückgabewert

Diese Funktion hat keinen Rückgabewert.

Anmerkung

- Bevor diese Funktion verwendet werden kann, müssen `SL2_Init()` und `RC_Init()` aufgerufen werden.

Beispiel

Die Drehrichtung von Servo 0 ändern wenn AVR Key 0 gedrückt wird:

```
int main ()
{
    //proxy function for whole initialization
    Init();

    //move servo number 0 to angle 2000
    RC_SetGlobalServoMode(RC_SERVO_MODE_PHI_DIRECT);
    RC_SetServoPhiDirect (0, 2000);

    while(1)
    {
        //set inverse bit only when key 0 is pressed
        if (SL2_AvrGetKey(0) == SL2_KEY_NOT_PRESSED)
        {
            RC_SetServoRotation(0, RC_SERVO_REVERSE_OFF);
        }
        else
        {
            RC_SetServoRotation(0, RC_SERVO_REVERSE_ON);
        }
    }

    return 0;
}
```

9.8.14 RC_SetServoAttributesTable

```
void RC_SetServoAttributesTable([in] uint8_t ServoNumber,
                                [in] RC_ServoAttributes *ServoAttr)
```

Beschreibung

Setzt die angegebenen Attribute für das angegebene Servo.

Parameter	Bedeutung
ServoNumber	Nummer des Servos, dessen Attribute verändert werden. Die Nummerierung der Servos ist in Abb. 7.1 auf Seite 81 ersichtlich.
ServoAttr	Datensatz der alle Eigenschaften eines Servos beinhaltet (siehe Abschnitt 9.11).

Rückgabewert

Diese Funktion hat keinen Rückgabewert.

Anmerkung

- Bevor diese Funktion verwendet werden kann, müssen `SL2_Init()` und `RC_Init()` aufgerufen werden.
- Statt dieser Funktion sollte `RC_SetGlobalServoAttributesTable`, da es keinen Sinn macht nur einzelne Servos zu kalibrieren.

9.8.15 RC_SetServoAttributesTable_P

Gleiche Funktion wie `RC_SetServoAttributesTable()`, die Tabelle muss jedoch im Programmspeicher abgelegt werden. Siehe dazu, „avr-libc“ Manual Kapitel „Program Space Utilities“.

9.8.16 RC_SetServoAttributesTable_E

Gleiche Funktion wie `RC_SetServoAttributesTable()`, die Tabelle muss jedoch im EEPROM abgelegt werden. Siehe dazu, „avr-libc“ Manual Kapitel „EEPROM handling“.

9.8.17 RC_SetGlobalServoAttributesTable

```
void RC_SetGlobalServoAttributesTable([in] RC_ServoAttributes)
```

Beschreibung

Setzt die angegebenen Attribute für alle Servos.

Parameter	Bedeutung
ServoAttributes	Feld von `RC_ServoAttributes` Datensätzen. Für jedes Servo muss ein Datensatz im Feld enthalten sein (siehe Abschnitt 9.11).

Rückgabewert

Diese Funktion hat keinen Rückgabewert.

Anmerkung

- Bevor diese Funktion verwendet werden kann, müssen `SL2_Init()` und `RC_Init()` aufgerufen werden.

Beispiel

Kopiert die Tabelle mit den Werten der Servolinearisierung, dem Offsetwinkel und dem Reverse Bit in den RoboticsChip und fährt die Mittelposition aller Servos an:

```
int main ()
{
    uint8_t i = 0;

    //proxy function for whole initialization
    Init();

    //copy the linearisation table into the RoboticsChip
    RC_SetGlobalServoAttributesTable_P(gTable);

    //activate all servos
    for (i=0; i<RC_NUMBER_OF_SERVOS; ++i)
    {
        RC_SetServoPhiDirect (i, 1609);
    }

    while(1)
    {
    }

    return 0;
}
```

9.8.18 RC_SetGlobalServoAttributesTable_P

Gleiche Funktion wie `RC_SetGlobalServoAttributesTable()`, die Tabelle muss jedoch im Programmspeicher abgelegt werden. Siehe dazu, „avr-libc“ Manual Kapitel „Program Space Utilities“.

9.8.19 RC_SetGlobalServoAttributesTable_E

Gleiche Funktion wie `RC_SetGlobalServoAttributesTable()`, die Tabelle muss jedoch im EEPROM abgelegt werden. Siehe dazu, „avr-libc“ Manual Kapitel „EEPROM handling“.

9.8.20 RC_SetServoTuneFine

```
inline void RC_SetServoTuneFine([in] int16_t Steps)
```

Beschreibung

Ermöglicht die Veränderung der Pulswiederholungszeit der PWM Signale (siehe Abschnitt 7.5.5 auf Seite 96 für eine genauere Beschreibung).

Parameter	Bedeutung
Steps	Anzahl der 0,375 μs Zyklen. Die Länge des Pulswiederholungszyklus ist von diesem Wert abhängig.

Rückgabewert

Diese Funktion hat keinen Rückgabewert.

Anmerkung

- Diese Funktion ist bei der Programmierung von Laufmustern nicht erforderlich
- Bevor diese Funktion verwendet werden kann, müssen `SL2_Init()` und `RC_Init()` aufgerufen werden.

Beispiel

siehe Beispiel der Funktion `RC_SetServoTuneCoarse()`.

9.8.21 RC_SetServoTuneCoarse

```
inline void RC_SetServoTuneCoarse([in] int16_t Steps)
```

Beschreibung

Ermöglicht die Veränderung der Pulswiederholungszeit der PWM Signale (siehe Abschnitt 7.5.5 auf Seite 96 für eine genauere Beschreibung).

Parameter	Bedeutung
Steps	Anzahl der 2,625 μs Zyklen. Die Länge des Pulswiederholungszyklus ist von diesem Wert abhängig.

Rückgabewert

Diese Funktion hat keinen Rückgabewert.

Anmerkung

- Diese Funktion ist bei der Programmierung von Laufmustern nicht erforderlich
- Bevor diese Funktion verwendet werden kann, müssen `SL2_Init()` und `RC_Init()` aufgerufen werden.

Beispiel

Die Pulswiederholungszeit wird auf ihren Initialwert gesetzt.

```
int main ()
{
    //proxy function for whole initialization
    Init();

    //set the impulse time to its default values
    RC_SetServoTuneFine(619);
    RC_SetServoTuneCoarse(619);

    while(1)
    {
    }
    return 0;
}
```

9.8.22 RC_GetPs2Data

`void RC_GetPs2Data([in] RC_Ps2Data *Data)`

Beschreibung

Liest die aktuellen Posititionsdaten der PS/2 Maus aus dem RoboticsChip und speichert diese in der der Funktion übergebenen Datenstruktur ab.

Parameter	Bedeutung
Data	Beinhaltet die Posititionsdaten der PS/2 Maus. Die Datenstruktur `RC_Ps2Data` ist in Abschnitt 9.11 auf Seite 216 erklärt.

Rückgabewert

Diese Funktion hat keinen Rückgabewert.

Anmerkung

Bevor diese Funktion verwendet werden kann, müssen `SL2_Init()` und `RC_Init()` aufgerufen werden.

Beispiel

Die Positionsdaten auf den Servo LEDs 0 und 1 ausgeben.

```
int main ()
{
    RC_Ps2Data MouseData;

    //proxy function for whole initialization
    Init();
```

```
    //clear all LEDs
    RC_SetLedClearAll();

    while(1)
    {
        RC_GetPs2Data (&MouseData);

        //show DeltaX and DeltaY on servo LEDs 0 and 1
        //clicks are shown on the FPGA message LEDs
        RC_SetLedBrightness(0, MouseData.DeltaX);
        RC_SetLedBrightness(1, MouseData.DeltaY);
    }

    return 0;
}
```

9.8.23 RC_SetRxControl

```
uint8_t RC_SetRxControl([out] uint8_t InvertPpm,
                        [out] uint8_t NumberOfChannels)
```

Beschreibung

Stellt die Empfangseinheit im RoboticsChip auf den verwendeten Fernsteuerungsempfänger ein.

Parameter	Bedeutung
InvertPpm	Bietet die Möglichkeit das PPM-Signal zu invertiert. Dieser Parameter ist nötig um auch Fernsteuerungen, die ein invertiertes Signal senden, verwenden zu können. Mögliche Werte: `RC_RX_NO_INVERT`, `RC_RX_INVERT`
NumberOfChannels	Anzahl der verwendeten Kanäle.

Rückgabewert

Rückgabewert	Beschreibung
RC_SUCCESSFUL	Empfangseinheit im RoboticsChip wurde kon guriert.
RC_INVALID_PARAMETER	Es wurde ein ungültiger Parameter übergeben, die Empfangseinheit im RoboticsChip wurde nicht kon guriert.

Anmerkung

- Bei den mitgelieferten Fernsteuerungen (Acoms Techniplus 40MHz AM 2/2/1/B) muss `InvertPpm` auf `RC_RX_NO_INVERT` und `NumberOfChannels` auf 2 gesetzt werden.
- Bevor diese Funktion verwendet werden kann, müssen `SL2_Init()` und `RC_Init()` aufgerufen werden.

Beispiel

siehe Beispiele der folgenden Funktion.

9.8.24 RC_GetRxData

`uint16_t RC_GetRxData([in] uint8_t ChannelNumber)`

Beschreibung

Liest die Daten von einem Kanal der Fernsteuerung.

Parameter	Bedeutung
ChannelNumber	Nummer des Kanals, der gelesen werden soll.

Rückgabewert

Rückgabewert	Beschreibung
Data	Von der Fernsteuerung gelesene Daten des angegebenen Kanals.

Anmerkung

Bevor diese Funktion verwendet werden kann, müssen `SL2_Init()` und `RC_Init()` aufgerufen werden.

Beispiel

Zeigt die empfangenen Daten direkt auf den Servo LEDs 0 und 1.

```
int main ()
{
    uint8_t Temp=0;
    uint16_t RxData=0;

    //proxy function for whole initialization
    Init();

    //clear all LEDs
    RC_SetLedClearAll();

    RC_SetRxControl (RC_RX_NO_INVERT, 2);

    while(1)
    {
        for (Temp=0; Temp<2; Temp++)
        {
            RxData = RC_GetRxData (Temp);
            if (RxData != 0)
            {
                //divide the number by 16, otherwise
                //changes are not visible (non-linear)
                RC_SetLedBrightness (Temp, (RxData >> 4));
            }
        }
    }

    return 0;
```

```
}
```

9.8.25 RC_GetGamePadButtons

`uint16_t RC_GetGamePadButtons()`

Beschreibung

Liest den Status aller Tasten des Sony Playstation 2 Gamepads.

Rückgabewert

Rückgabewert	Beschreibung
Buttons	Am Gamepad gedrückte Tasten. Die Zuordnung zwischen Bits und Tasten entspricht jener in Abbildung 7.29 auf Seite 121.

Anmerkung

Bevor diese Funktion verwendet werden kann, müssen `SL2_Init()` und `RC_Init()` aufgerufen werden.

Beispiel

Deaktiviert Servo LED 0, wenn eine der Richtungstasten am Gamepad gedrückt wird.

```
int main ()
{
    uint16_t Data;

    //proxy function for whole initialization
    Init();

    //clear all LEDs again
    RC_SetLedClearAll();

    while(1)
    {
        Data = RC_GetGamePadButtons();

        //ignore all other buttons
        Data = Data >> 4;
        Data = Data & 0x000F;

        if (Data == 0x0F)
        {
            //switch LED on
            RC_SetLedBrightness(0, 255);
        }
        else
        {
            //clear LED
```

```
            RC_SetLedBrightness(0, 0);
        }
    }
    return 0;
}
```

9.8.26 RC_GetGamePadJoystickData

```
void RC_GetGamePadJoystickData([in] uint8_t Position,
                               [out] RC_GamePadJoystickData *Data)
```

Beschreibung

Liest die aktuelle Stellung der Joysticks am Sony Playstation 2 Gamepad aus dem RoboticsChip.

Parameter	Bedeutung
Position	Gibt an, ob der linke oder rechte Joystick verwendet wird. Dieser Parameter kann den Wert `RC_GAME_PAD_LEFT_JOY` oder `RC_GAME_PAD_RIGHT_JOY` annehmen.
Data	Beinhaltet die Stellung beider Joysticks. Der Aufbau dieser Struktur ist in Abschnitt 9.11 auf Seite 216 erklärt.

Rückgabewert

Diese Funktion hat keinen Rückgabewert.

Anmerkung

Bevor diese Funktion verwendet werden kann, müssen `SL2_Init()` und `RC_Init()` aufgerufen werden.

Beispiel

Die Bewegungen des linken Joysticks an den Servo LEDs 0 und 1 anzeigen:

```
int main ()
{
    RC_GamePadJoystickData Data;

    //proxy function for whole initialization
    Init();

    //clear all LEDs again
    RC_SetLedClearAll();

    while(1)
    {
        RC_GetGamePadJoystickData (RC_GAME_PAD_LEFT_JOY, &Data);

        RC_SetLedBrightness (0, Data.X);
        RC_SetLedBrightness (1, Data.Y);
```

```
    }

    return 0;
}
```

9.8.27 RC_GetGamePadData

`void RC_GetGamePadData ([out] RC_GamePadData *Data)`

Beschreibung

Liest die aktuelle Stellung der Joysticks und die Tasten des Sony Playstation 2 Gamepad aus dem RoboticsChip.

Parameter	Bedeutung
Data	Beinhaltet die Stellung beider Joysticks und den Status der Tasten. Der Aufbau dieser Struktur ist in Abschnitt 9.11 auf Seite 216 erklärt.

Rückgabewert

Diese Funktion hat keinen Rückgabewert.

Anmerkung

Bevor diese Funktion verwendet werden kann, müssen `SL2_Init()` und `RC_Init()` aufgerufen werden.

Beispiel

Die Bewegungen der Joysticks an den Servo LEDs 0 – 3 anzeigen:

```
int main ()
{
    RC_GamePadData GameData;

    //proxy function for whole initialization
    Init();

    //clear all LEDs again
    SetLedClearAll();

    while(1)
    {
        RC_GetGamePadData(&GameData);

        RC_SetLedBrightness (0, GameData.JoystickLeft.X);
        RC_SetLedBrightness (1, GameData.JoystickLeft.Y);
        RC_SetLedBrightness (2, GameData.JoystickRight.X);
        RC_SetLedBrightness (3, GameData.JoystickRight.Y);
    }

    return 0;
```

```
}
```

9.8.28 RC_SetBuzzerVolume

```
uint8_t RC_SetBuzzerVolume ([in] uint8_t Volume)
```

Beschreibung

Bestimmt die Lautstärke des Piezo-Summers.

Parameter	Bedeutung
Volume	Lautstärke des Piezo-Summers. Die Lautstärke ist in drei Stufen einstellbar: mittlere Lautstärke (`RC_BUZZER_VOL_MIDDLE`), hohe Lautstärke (`RC_BUZZER_VOL_HIGH`) und ausgeschaltet (`RC_BUZZER_VOL_OFF`).

Rückgabewert

Rückgabewert	Beschreibung
Status	`RC_INVALID_PARAMETER` wenn der Funktion ungültige Parameter übergeben wurden, sonst `RC_SUCCESSFUL`.

Anmerkung

Bevor diese Funktion verwendet werden kann, müssen `SL2_Init()` und `RC_Init()` aufgerufen werden.

Beispiel

Siehe folgende Funktionen.

9.8.29 RC_SetBuzzerPhaseHigh

```
void RC_SetBuzzerPhaseHigh ([in] uint16_t PhaseHigh)
```

Beschreibung

Bestimmt die Länge der High-Phase des Piezo-Summer Signals (Rechtecksignal). Die Frequenz dieses Signals kann wie folgt berechnet werden:

$$f_{rect} = \frac{1}{(T_{PhaseHigh} + T_{PhaseLow}) \cdot 125\ ns}$$

Parameter	Bedeutung
PhaseHigh	Länge der High-Phase des Piezo-Summer Signals.

Rückgabewert

Diese Funktion hat keinen Rückgabewert.

Anmerkung

Bevor diese Funktion verwendet werden kann, müssen `SL2_Init()` und `RC_Init()` aufgerufen werden.

Beispiel

Siehe folgende Funktion.

9.8.30 RC_SetBuzzerPhaseLow

`void RC_SetBuzzerPhaseLow ([in] uint16_t PhaseLow)`

Beschreibung

Bestimmt die Länge der Low-Phase des Piezo-Summer Signals (Rechtecksignal). Die Frequenz dieses Signals kann wie folgt berechnet werden:

$$f_{rect} = \frac{1}{(T_{PhaseHigh} + T_{PhaseLow}) \cdot 125\ ns}$$

Parameter	Bedeutung
PhaseLow	Länge der Low-Phase des Piezo-Summer Signals.

Rückgabewert

Diese Funktion hat keinen Rückgabewert.

Anmerkung

Bevor diese Funktion verwendet werden kann, müssen `SL2_Init()` und `RC_Init()` aufgerufen werden.

Beispiel

Erzeugt ein Rechtecksignal mit einer Frequenz von 100 Hz für den Summer und schaltet diesen auf maximale Lautstärke:

```
int main ()
{
    //proxy function for whole initialization
    Init();

    //clear all LEDs again
    RC_SetLedClearAll();

    //generate a 100Hz oscillation
    //2*40000*125ns = 0.01s
    RC_SetBuzzerPhaseHigh (40000);
    RC_SetBuzzerPhaseLow (40000);
```

```
    //activate the buzzer, high volume
    RC_SetBuzzerVolume(RC_BUZZER_VOL_HIGH);


    //do nothing
    while(1)
    {
    }

    return 0;
}
```

9.8.31 RC_SetCordicCalculationArguments

```
uint8_t RC_SetCordicCalculationArguments([in] uint16_t RegisterX,
                                         [in] uint16_t RegisterY,
                                         [in] uint16_t RegisterZ,
                                         [in] uint8_t Mode,
                                         [in] uint8_t CoordSys)
```

Beschreibung

Mit dieser Funktion wird eine Berechnung mit dem CORDIC-Algorithmus eingeleitet. Zunächst werden die Parameter mittels der Funktion `RC_SetCordicCalculationArguments()` in den RoboticsChip geschrieben, die Berechnung mit `RC_SetCordicCalculationStart()` gestartet und die Ergebnisse mit `RC_GetCordicCalculationResult()` gelesen nachdem mindestens so lange gewartet wurde, bis `RC_CordicWaitOnResult()` nicht mehr blockiert.
Eine genaue Beschreibung der Möglichkeiten des CORDIC-Algorithmus be ndet sich in Abschnitt 7.22.
Diese Funktion führt die über die Parameter Mode und CoordSys bestimmte Funktion mit den Parametern RegisterX, RegisterY und RegisterZ aus.

Parameter	Bedeutung
RegisterX	Initialwert des X-Registers.
RegisterY	Initialwert des Y-Registers.
RegisterZ	Initialwert des Z-Registers.
Mode	Modus der Berechnung. Dieser Parameter kann die Werte RC_CORDIC_MODE_ROTATION bzw. RC_CORDIC_MODE_VECTORING annehmen. Siehe Abschnitt 7.22.
CoordSys	Bestimmt das für die Berechnung verwendete Koordinatensystem (RC_CORDIC_COORD_LIN, RC_CORDIC_COORD_CIR oder RC_CORDIC_COORD_HYP). Siehe Abschnitt 7.22.

Rückgabewert

Rückgabewert	Beschreibung
RC_FAILED	Die Queue des RoboticsChips ist bereits gänzlich befüllt.
RC_SUCCESSFUL	Die Berechnung wurde in den Speicher des RoboticsChips geschrieben.

Anmerkung

- Um sinnvolle Ergebnisse zu erhalten, müssen die in Abschnitt 7.22 beschriebenen Zahlenformate und Konvergenzbedingungen beachtet werden.
- Bei Verwendung des hyperbolischen Koordinatensystems ist die Verwendung des Rotationsmodus nicht möglich (vgl. Abschnitt 7.22).
- Mittels dieser Funktion werden die Parameter nur in den dafür vorgesehenen Speicherbereich des RoboticsChips kopiert, die Berechnung wird dadurch noch nicht gestartet.
- Bevor diese Funktion verwendet werden kann, müssen `SL2_Init()` und `RC_Init()` aufgerufen werden und der CORDIC-Interrupt muss aktiviert sein.

Beispiel

Siehe Beispiele in Abschnitt 9.8.34 auf Seite 199.

9.8.32 RC_SetCordicCalculationStart

```
void RC_SetCordicCalculationStart ()
```

Beschreibung

Startet die CORDIC-Berechnungen jener Werte, die zuvor mittels `RC_SetCordicCalculation-Arguments()` in die Queue geschrieben wurden.

Rückgabewert

Diese Funktion hat keinen Rückgabewert.

Anmerkung

Bevor diese Funktion verwendet werden kann, müssen `SL2_Init()` und `RC_Init()` aufgerufen werden und der CORDIC-Interrupt muss aktiviert sein.

Beispiel

Siehe Beispiele in Abschnitt 9.8.34 auf Seite 199.

9.8.33 RC_CordicWaitOnResult

```
void RC_CordicWaitOnResult ()
```

Beschreibung

Wartet so lange, bis der CORDIC-Interrupt vom RoboticsChip ausgelöst wird und somit das Ergebnis verfügbar ist.

Rückgabewert

Diese Funktion hat keinen Rückgabewert.

Anmerkung

Bevor diese Funktion verwendet werden kann, müssen `SL2_Init()` und `RC_Init()` aufgerufen werden und der CORDIC-Interrupt muss aktiviert sein.

Beispiel

Siehe Beispiele in Abschnitt 9.8.34 auf Seite 199.

9.8.34 RC_GetCordicCalculationResult

```
uint16_t RC_GetCordicCalculationResult(
              [in] uint8_t Register,
              [in] uint8_t NumberOfCalculation)
```

Beschreibung

Liest das Ergebnis einer CORDIC-Berechnung aus dem RoboticsChip.

Parameter	Bedeutung
Register	Auswahl des Registers aus dem gelesen werden soll (kann die Werte `'x'`, `'X'`, `'y'`, `'Y'`, `'z'` oder `'Z'` annehmen).
NumberOfCalculation	Index der Berechnung in der Queue.

Rückgabewert

Rückgabewert	Beschreibung
Result	Ergebnis der CORDIC-Berechnung.

Anmerkung

Bevor diese Funktion verwendet werden kann, müssen `SL2_Init()` und `RC_Init()` aufgerufen werden und der CORDIC-Interrupt muss aktiviert sein.

Beispiel

Berechnet den Arcustangens von π:

```
int main (void)
{
    uint16_t Result = 0;

    // initialize all modules
    Init();

    while (1)
    {
```

```
        // calculate atan(pi) by using the CORDIC algorithm
        // in this mode, the CORDIC algorithm calculates atan(y/x), so
        // y will be set to pi and x will be set to 1
        // for a higher precision of the argument, both values are multiplied
        // by 4096 (shift left by 12 bits)
        // write the parameters into the RoboticsChip and start the calculation
        RC_SetCordicCalculationArguments (4096, 12868, 0, RC_CORDIC_MODE_VECTORING
            , RC_CORDIC_COORD_CIR);
        RC_SetCordicCalculationStart ();

        // wait until the calculation is done and read the result
        RC_CordicWaitOnResult ();
        Result = (RC_GetCordicCalculationResult('z', 0));

        // Result will have the value 20687 here, which is equal to
        // atan(pi)*(2^14)
    }

    return 0;
}
```

9.8.35 RC_GetRoboticsChipVersion

`uint16_t RC_GetRoboticsChipVersion ()`

Beschreibung

Liest die Version der Kon guration des RoboticsChips.

Rückgabewert

Rückgabewert	Beschreibung
Version	Versionsnummer als Dezimalzahl.

Anmerkung

Bevor diese Funktion verwendet werden kann, müssen `SL2_Init()` und `RC_Init()` aufgerufen werden und der CORDIC-Interrupt muss aktiviert sein.

9.8.36 RC_GetGlobalTimer

`uint16_t RC_GetGlobalTimer ()`

Beschreibung

Liest die Anzahl der seit dem Einschalten vergangenen 20 ms Takte des RoboticsChips.

Rückgabewert

Rückgabewert	Beschreibung
Result	Anzahl der 20 ms Takte des RoboticsChips.

Anmerkung

Bevor diese Funktion verwendet werden kann, müssen `SL2_Init()` und `RC_Init()` aufgerufen werden.

Beispiel

Schaltet LED 0 nach 100s ein.

```
int main (void)
{
    uint16_t Timer = 0;

    //proxy function for whole initialization
    Init();

    for(;;)
    {
    // switch on servo LED 0 after 100s
        Timer = RC_GetGlobalTimer();
        if (Timer > 5000)
        {
            RC_SetLedBrightness(0, 255);
        }
    }

    return 0;
}
```

9.8.37 RC_SetInterrupt0

`void RC_SetInterrupt0 (uint16_t Interrupts)`

Beschreibung

Schreibt die Interrupt Maske für Interrupt 0 in den Robotics Chip.
Über diese Funktion können die Interrupts für Paniktaste, CORDIC, ... aktiviert werden.

Parameter	Bedeutung
Interrupts	Interrupt Maske von Interrupt 0. Die Interrupt-Mask-Bits nden sich in den in den Abschnitten 7.5.3.1, 7.13.1.1 und 7.22 beschrieben.

Rückgabewert

Diese Funktion hat keinen Rückgabewert.

Anmerkung

Bevor diese Funktion verwendet werden kann, müssen `SL2_Init()` und `RC_Init()` aufgerufen werden.

Beispiel

Einschalten der Interrupts für CORDIC und Paniktaste:

```
int main (void)
{
    uint8_t i = 0;

    //proxy function for whole initialization
    Init();

    // copy the linearisation table into the RoboticsChip
    RC_SetGlobalServoAttributesTable_P(gTable);

    //activate all servos
    for (i=0; i<RC_NUMBER_OF_SERVOS; ++i)
    {
        RC_SetServoPhiDirect (i, 1609);
    }
    RC_SetGlobalServoMode(RC_SERVO_MODE_PHI_DIRECT);

    // enable cordic interrupt
    RC_SetInterrupt0(RC_INTTERRUPT_MASK_CORDIC+
                     RC_INTTERRUPT_MASK_PANIC);

    for(;;);

    return 0;
}
```

9.9 Inverse Kinematik

Mit Hilfe der inversen Kinematik ist es möglich, die Winkel aller Gelenke (Servos) aus einem gegeben Fußpunkt zu berechnen. Weitere Informationen sind in Kapitel 8 zu nden.

9.9.1 IK_Init

```
void IK_Init()
```

Beschreibung

Initialisiert das inverse Kinematik Modul.

Rückgabewert

Diese Funktion hat keinen Rückgabewert.

Anmerkung

- Muss vor allen anderen Funktionen des Moduls aufgerufen werden.

9.9.2 IK_CalculateInverseKinematicsCoxa

```
IK_InverseKinematicsReturn CalculateInverseKinematicsCoxa(
      [in] int16_t XCoord,
      [in] int16_t YCoord,
      [in] int16_t ZCoord,
      [in] int8_t Position)
```

Beschreibung

Berechnet die inverse Kinematik eines Beines.
In dieser Berechnung werden die Einschränkungen durch die Mechanik des Hexapods nicht berücksichtigt.

Parameter	Bedeutung
XCoord	X-Koordinate des gewünschten Fußpunktes.
YCoord	Y-Koordinate des gewünschten Fußpunktes.
ZCoord	Z-Koordinate des gewünschten Fußpunktes.
Position	Da es für viele Punkte zwei verschiedene Lösungen für φ_1 und φ_2 gibt muss zusätzlich angegeben werden, welche der beiden Lösungen berechnet werden soll. Wird dieser Parameter `IK_KNEE_UP` gesetzt, so wird eine Lösung berechnet, in welcher der Fußpunkt stets eine tiefere Position einnimmt als das „Kniegelenk“. Bei `IK_KNEE_DOWN` liegt der Fußpunkt höher als das „Kniegelenk“.

Rückgabewert

Der Rückgabewert vom Typ `IK_InverseKinematicsReturn` ist eine Struktur, welche die berechneten Punkte beeinhaltet sowie Informationen ob die Berechnung erfolgreich war. Eine Berechnung ist dann nicht erfolgreich, wenn über die Parameter `XCoord`, `YCoord` und `ZCoord` ein Punkt angegeben wird, der von der Beinmechanik nicht erreicht werden kann.
Der Aufbau der Struktur `IK_InverseKinematicsReturn` ist in Abschnitt 9.11 auf Seite 216 genauer erklärt.

Rückgabewert	Beschreibung
IK_FAILED	Der Punkt kann mit den gegebenen Längen der Beine nicht erreicht werden, die Rückgabewerte der Winkel sind ungültig und dürfen nicht angefahren werden.
IK_SUCCESSFUL	Alle Winkel wurden erfolgreich berechnet.

Anmerkung

- Auch wenn die Berechnung eines Winkels erfolgreich war, darf dieser Winkel nicht ohne weiteres eingestellt werden. So könnte eine Berechnung z.B. einen Winkel von 60° für den Coxa-Servo (Hüft-Servo) ergeben, der maximale Winkel für dieses Servo ist allerdings durch die Mechanik auf 45° beschränkt.
  Bevor ein berechneter Winkel angefahren wird muss überprüft werden, ob dieser Winkel (ohne die Mechanik zu beschädigen) angefahren werden kann.

- Bevor diese Funktion verwendet werden kann, müssen `SL2_Init()`, `RC_Init()` und `RC_Init()` aufgerufen werden und der CORDIC-Interrupt muss aktiviert sein.

Beispiel

Berechnet die Winkel für Punkt (159, 0, 0) im Beinkoordinatensystem und fährt diesen mit allen drei Servos an:

```
int main (void)
{
    uint8_t i=0;
    IK_InverseKinematicsReturn Data;


    // initialize all modules
    Init();

    // copy the linearisation table into the RoboticsChip
    RC_SetGlobalServoAttributesTable_P(gTable);

    //activate all servos
    for (i=0; i<RC_NUMBER_OF_SERVOS; ++i)
    {
        RC_SetServoPhiDirect (i, 1609);
    }
    RC_SetGlobalServoMode(RC_SERVO_MODE_PHI_DIRECT);

    // enable cordic interrupt
    RC_SetInterrupt0(RC_INTTERRUPT_MASK_CORDIC);


    // calculate the angles needed to reach the desired point
    Data = IK_CalculateInverseKinematicsCoxa (159, 0, 0, IK_KNEE_UP);

    // return in case the point cannot be reached
    if (Data.Calculation == IK_SUCCESSFUL)
    {
        // normalize the calculated angles to the data format of the
        // RoboticsChip and add the PWM value for the reference angle
        Data.Angles.Phi0 = (uint16_t)((((int32_t)(Data.Angles.Phi0))>>3)+
            IK_PHI_0_COOR_SYS_OFFSET);
        Data.Angles.Phi1 = (uint16_t)((((int32_t)(Data.Angles.Phi1))>>3)+
            IK_PHI_1_COOR_SYS_OFFSET);
        Data.Angles.Phi2 = (uint16_t)((((int32_t)(Data.Angles.Phi2))>>3)+
            IK_PHI_2_COOR_SYS_OFFSET);

        // the mechanics do not allow following angles
        if (!((Data.Angles.Phi0 < IK_PHI_0_MIN) ||
              (Data.Angles.Phi0 > IK_PHI_0_MAX) ||
              (Data.Angles.Phi1 < IK_PHI_1_MIN) ||
              (Data.Angles.Phi1 > IK_PHI_1_MAX) ||
              (Data.Angles.Phi2 < IK_PHI_2_MIN) ||
              (Data.Angles.Phi2 > IK_PHI_2_MAX)))
        {
            // move the leg directly to the desired point and return
            RC_SetServoPhiDirect (0, Data.Angles.Phi0);
```

```
            RC_SetServoPhiDirect (1, Data.Angles.Phi1);
            RC_SetServoPhiDirect (2, Data.Angles.Phi2);
        }
    }


    for(;;);

    return 0;
}
```

9.9.3 IK_SetLegMovementCoxa

```
uint8_t IK_SetLegMovementCoxa([in] uint8_t LegNumber,
                              [in] int16_t XCoord,
                              [in] int16_t YCoord,
                              [in] int16_t ZCoord,
                              [in] int8_t Position)
```

Beschreibung

Berechnet die inverse Kinematik eines angegebenen Beines im Coxa-System und bewegt den Fußpunkt des Beines zu diesem. Kann der angegebene Punkt nicht angefahren werden, so wird das Bein nicht bewegt.

Parameter	Bedeutung
LegNumber	Index des Hexapod Beines, dieser Index entspricht jenem, der auf der Oberseite des Hexapods eingraviert ist.
XCoord	X-Koordinate des gewünschten Fußpunktes.
YCoord	Y-Koordinate des gewünschten Fußpunktes.
ZCoord	Z-Koordinate des gewünschten Fußpunktes.
Position	Da es für viele Punkte zwei verschiedene Lösungen für φ_1 und φ_2 gibt muss zusätzlich angegeben werden, welche der beiden Lösungen berechnet werden soll. Wird dieser Parameter `IK_KNEE_UP` gesetzt, so wird eine Lösung berechnet, in welcher der Fußpunkt stets eine tiefere Position einnimmt als das „Kniegelenk“. Bei `IK_KNEE_DOWN` liegt der Fußpunkt höher als das „Kniegelenk“.

Rückgabewert

Der Rückgabewert bestimmt, ob der Punkt angefahren wurde oder nicht.

Rückgabewert	Beschreibung
IK_FAILED	Der Punkt kann nicht erreicht werden.
IK_SUCCESSFUL	Der Punkt wurde angefahren.

Anmerkung

- Bevor diese Funktion verwendet werden kann, müssen `SL2_Init()`, `RC_Init()` und `RC_Init()` aufgerufen werden und der CORDIC-Interrupt muss aktiviert sein.

- Von dieser Funktion werden sowohl die Einschränkungen durch die Längen der Beinelemente als auch durch die maximalen Drehwinkel der Servos beachtet.

Beispiel

Berechnet die Winkel für Punkt (159, 0, 0) und fährt diesen an:

```
int main (void)
{
    uint8_t i=0;
    IK_InverseKinematicsReturn Data;


    // initialize all modules
    Init();

    // copy the linearisation table into the RoboticsChip
    RC_SetGlobalServoAttributesTable_P(gTable);

    //activate all servos
    for (i=0; i<RC_NUMBER_OF_SERVOS; ++i)
    {
        RC_SetServoPhiDirect (i, 1609);
    }
    RC_SetGlobalServoMode(RC_SERVO_MODE_PHI_DIRECT);

    // enable cordic interrupt
    RC_SetInterrupt0(RC_INTTERRUPT_MASK_CORDIC);

    // move leg 3 to (159, 0, 0)
    IK_SetLegMovementCoxa (3, 159, 0, 0, IK_KNEE_UP);


    for(;;);

    return 0;
}
```

9.9.4 IK_SetLegMovement

```
uint8_t IK_SetLegMovement([in] uint8_t LegNumber,
                          [in] int16_t XCoord,
                          [in] int16_t YCoord,
                          [in] int16_t ZCoord,
                          [in] int8_t Position)
```

Beschreibung

Berechnet die inverse Kinematik eines angegebenen Beines und bewegt den Fußpunkt des Beines zu diesem. Die angegebenen Koordinaten beziehen sich auf das rotierte Coxa-System. Das Coxa-System wurde so rotiert, dass die Achsen parallel zum Korpus-System sind. Kann der angegebene Punkt nicht angefahren werden, so wird das Bein nicht bewegt.

Parameter	Bedeutung
LegNumber	Index des Hexapod Beines, dieser Index entspricht jenem, der auf der Oberseite des Hexapods eingraviert ist.
XCoord	X-Koordinate des gewünschten Fußpunktes.
YCoord	Y-Koordinate des gewünschten Fußpunktes.
ZCoord	Z-Koordinate des gewünschten Fußpunktes.
Position	Da es für viele Punkte zwei verschiedene Lösungen für φ_1 und φ_2 gibt muss zusätzlich angegeben werden, welche der beiden Lösungen berechnet werden soll. Wird dieser Parameter `IK_KNEE_UP` gesetzt, so wird eine Lösung berechnet, in welcher der Fußpunkt stets eine tiefere Position einnimmt als das „Kniegelenk“. Bei `IK_KNEE_DOWN` liegt der Fußpunkt höher als das „Kniegelenk“.

Rückgabewert

Der Rückgabewert bestimmt, ob der Punkt angefahren werden konnte oder nicht.

Rückgabewert	Beschreibung
IK_FAILED	Der Punkt kann nicht erreicht werden.
IK_SUCCESSFUL	Der Punkt wurde angefahren.

Anmerkung

- Bevor diese Funktion verwendet werden kann, müssen `SL2_Init()`, `RC_Init()` und `RC_Init()` aufgerufen werden und der CORDIC-Interrupt muss aktiviert sein.

Beispiel

Berechnet die Winkel für den Punkt (138, 80, 0) im rotierten Coxa-System (dieser Punkt entspricht (159, 0, 0) im Coxa-System) und fährt diesen an:

```
//program entry point
int main (void)
{
    uint8_t i=0;


    // initialize all modules
    Init();

    // copy the linearisation table into the RoboticsChip
    RC_SetGlobalServoAttributesTable_P(gTable);

    //activate all servos
    for (i=0; i<RC_NUMBER_OF_SERVOS; ++i)
    {
        RC_SetServoPhiDirect (i, 1609);
    }
    RC_SetGlobalServoMode(RC_SERVO_MODE_PHI_DIRECT);

    // enable cordic interrupt
    RC_SetInterrupt0(RC_INTTERRUPT_MASK_CORDIC);
```

```
    //calculate the inverse kinematics
    IK_SetLegMovement (2, 138, 80, 0, IK_KNEE_UP);


    for(;;);

    return 0;
}
```

9.9.5 IK_GivensTransformation

```
IK_Point2d IK_GivensTransformation([in] IK_Point2d Input,
                                   [in] uint8_t Leg)
```

Beschreibung

Führt die Givens-Transformation für 2-dimensionale Punkte mit Vielfachen von 60° aus.

Parameter	Bedeutung
Input	Koordinaten des zu rotierenden Koordinatensystems.
Leg	Index des Beins, für das die Rotation ausgeführt werden soll. Der Index bestimmt den Winkel der Rotationaufgrund der Anordnung des Beins am Körper.

Rückgabewert

Der Rückgabewert bestimmt, ob der Punkt angefahren wurde oder nicht.

Rückgabewert	Beschreibung
Point	Koordinaten des Punktes im rotierten Koordinatensystem.

Anmerkung

- Bevor diese Funktion verwendet werden kann, müssen `SL2_Init()`, `RC_Init()` und `RC_Init()` aufgerufen werden und der CORDIC-Interrupt muss aktiviert sein.

Beispiel

Berechnet die Winkel für den Punkt (138, 80, 0) im rotierten Coxa-System (dieser Punkt entspricht (159, 0, 0) im Coxa-System) und fährt diesen an:

```
int main (void)
{
    uint8_t i=0;
    IK_Point2d Coords;


    // initialize all modules
    Init();

    // copy the linearisation table into the RoboticsChip
    RC_SetGlobalServoAttributesTable_P(gTable);

    //activate all servos
```

```
    for (i=0; i<RC_NUMBER_OF_SERVOS; ++i)
    {
        RC_SetServoPhiDirect (i, 1609);
    }
    RC_SetGlobalServoMode(RC_SERVO_MODE_PHI_DIRECT);

    // enable cordic interrupt
    RC_SetInterrupt0(RC_INTTERRUPT_MASK_CORDIC);

    // points in the rotated coxa system
    Coords.x = 138;
    Coords.y = 80;


    // rotate the displaced corpus system to the coxa system
    Coords = IK_GivensTransformation (Coords, 2);

    //calculate the inverse kinematics
    IK_SetLegMovementCoxa (2, Coords.x, Coords.y, 0, IK_KNEE_UP);


    for(;;);

    return 0;
}
```

9.10 SD-Karte

9.10.1 mmc_init

`uint8_t mmc_init()`

Beschreibung

Initialisiert die SD-Karte.

Rückgabewert

Rückgabewert	Beschreibung
FAILED	Die Initialisierung ist fehlgeschlagen.
SUCCESSFUL	Die Funktion wurde erfolgreich ausgeführt.

Anmerkung

- Die SD-Karte muss eingelegt sein.
- Bevor diese Funktion verwendet werden kann, muss `SL2_Init()` aufgerufen werden.

Beispiel

Siehe Beispiele der folgenden Funktionen.

9.10.2 fat_cluster_data_store

`void fat_cluster_data_store()`

Beschreibung

Liest Kon gurationsdaten wie Clustergröße, Anzahl der Bytes pro Cluster, ... von der SD-Karte und legt diese in einer Datenstruktur ab.
Diese Datenstruktur wird nur intern verwendet, die darin enthaltenen Daten können über die Funktionen `GetClusterSize()` und `GetBytesPerSec()` ausgelesen werden.

Rückgabewert

Diese Funktion hat keinen Rückgabewert.

Anmerkung

- Die SD-Karte muss eingelegt und initialisiert sein (mittels `mmc_init()`).
- Bevor diese Funktion verwendet werden kann, muss `SL2_Init()` aufgerufen werden.

Beispiel

siehe Beispiel in Abschnitt 9.10.4 auf Seite 211.

9.10.3 GetClusterSize

`uint8_t GetClusterSize()`

Beschreibung

Gibt die Clustergröße zurück, welche zuvor durch die Funktion `fat_cluster_data_store()` ermittelt wurde.

Rückgabewert

Rückgabewert	Beschreibung
ClusterSize	Clustergröße.

Anmerkung

- Die SD-Karte muss eingelegt und initialisiert sein.
- Die Funktion `fat_cluster_data_store()` muss zuvor aufgerufen worden sein.
- Bevor diese Funktion verwendet werden kann, muss `SL2_Init()` aufgerufen werden.

Beispiel

siehe Beispiel in Abschnitt 9.10.4 auf Seite 211.

9.10.4 GetBytesPerSec

`uint16_t GetBytesPerSec()`

Beschreibung

Gibt die Anzahl der Bytes pro Cluster zurück, welche zuvor durch die Funktion `fat_cluster_data_store()` ermittelt wurde.

Rückgabewert

Rückgabewert	Beschreibung
BytesPerSec	Anzahl der Bytes pro Sektor.

Anmerkung

- Die SD-Karte muss eingelegt und initialisiert sein.
- Die Funktion `fat_cluster_data_store()` muss zuvor aufgerufen worden sein.
- Bevor diese Funktion verwendet werden kann, muss `SL2_Init()` aufgerufen werden.

Beispiel

Ermittelt die Clustergröße sowie die Anzahl der Bytes pro Cluster der eingelegten SD-Karte und sendet diese über die USB-Schnittstelle an den PC.

```
int main ()
{
    uint8_t Temp=0;
    uint8_t ClusterSize = 0;
    uint16_t BytesPerSec = 0;

    //function that has to be executed in every appli-
    //cation to initialize the interface with the FPGA
    InitSandboxS2();

    //initialize the USB port to send debug information
    //to the PC
    TextInterfaceInit(38400);


    //initialize the memory card
    //this has to be done here AND before the memory card
    //is used because otherwise there occur problems with
    //various cards at power on
    do
    {
        Temp = mmc_init();
    } while ( Temp != SUCCESSFUL);

    //clear all LEDs again
    SetLedClearAll();
```

```
    //wait until the key is released and pressed again
    AvrWaitOnKey (KEY_1, KEY_NOT_PRESSED);
    AvrWaitOnKey (KEY_1, KEY_PRESSED);


    //initialize the memory card again
    do
    {
        Temp = mmc_init();
    } while ( Temp != SUCCESSFUL);

    printf("accessing␣memory␣card...\n");

    //save cluster OFFSET und size
    fat_cluster_data_store();

    //get cluster size and bytes per sector
    ClusterSize = GetClusterSize();
    BytesPerSec = GetBytesPerSec();

    //print some more data on the screen
    printf("reading␣cluster␣size:␣%d\n", ClusterSize);
    printf("reading␣bytes␣per␣sector:␣%u\n", BytesPerSec);


    while(1)
    {
    }

    return 0;
}
```

9.10.5 fat_search_file

```
uint8_t fat_search_file ([inout] uint8_t *File_Name,
                         [inout] uint16_t *Cluster,
                         [out] uint32_t *Size,
                         [out] uint8_t *Dir_Attrib,
                         [inout] uint8_t *Buffer)
```

Beschreibung

Sucht die Datei mit dem angegebenen Namen.

Parameter	Bedeutung
File_Name	Startadresse des Strings für den Dateinamen.
Cluster	Der Verzeichnis-Cluster in dem gesucht werden soll. Dieser Parameter gibt die Adresse des ersten Clusters der Datei zurück wenn diese gefunden wurde.
Size	Enthält die Größe der Datei, sofern diese gefunden wurde.
Dir_Attrib	Attribute des Verzeichnisses.
Buffer	Zwischenspeicher

Rückgabewert

Rückgabewert	Beschreibung
SUCCESSFUL	Die Datei wurde gefunden.
FILE_NOT_FOUND	Die Datei wurde nicht gefunden.
FAILED	Mindestens einer der angegebenen Parameter ist ungültig oder die SD-Karte wurde nicht korrekt initialisiert.

Anmerkung

- Die SD-Karte muss eingelegt und initialisiert sein.
- Wird der Parameter `Cluster` auf 0 gesetzt, so wird im Root-Directory gesucht.
- Bevor diese Funktion verwendet werden kann, muss `SL2_Init()` aufgerufen werden.

Beispiel

siehe Beispiel in Abschnitt 9.10.7 auf Seite 214.

9.10.6 fat_read_file

```
void fat_read_file ([in] uint16_t Cluster,
                    [out] uint8_t *Buffer,
                    [in] uint32_t BlockCount)
```

Beschreibung

Liest Daten aus einer Datei.

Parameter	Bedeutung
Cluster	Erster Cluster der Datei.
Buffer	Gelesene Daten; Buffer ist ein eindimensionales Array (8-bit Datentyp), welches mindestens so groß ist wie die Anzahl der Bytes pro Sektor.
BlockCount	Zu lesender Teil der Datei. Da eine Datei gösser sein kann als ein einziger Cluster, muss angegeben werden aus dem welchem Cluster der Datei gelesen werden soll.

Rückgabewert

Diese Funktion hat keinen Rückgabewert.

Anmerkung

- Die SD-Karte muss eingelegt und initialisiert sein.
- Bevor diese Funktion verwendet werden kann, muss `SL2_Init()` aufgerufen werden.

Beispiel

siehe Beispiel in Abschnitt 9.10.7 auf Seite 214.

9.10.7 fat_write_file

```
void fat_write_file ([in] uint16_t cluster,
                     [in] uint8_t *buffer,
                     [in] uint32_t blockCount)
```

Beschreibung

Schreibt Daten in eine Datei.

Parameter	Bedeutung
Cluster	Erster Cluster der Datei.
Buffer	Zu schreibende Daten; Buffer ist ein eindimensionales Array (8-bit Datentyp), welches mindestens so groß ist wie die Anzahl der Bytes pro Sektor.
BlockCount	Zu schreibender Teil der Datei. Da eine Datei gösser sein kann als ein einziger Cluster, muss angegeben werden in welchem Cluster der Datei geschrieben werden soll.

Rückgabewert

Diese Funktion hat keinen Rückgabewert.

Anmerkung

- Die SD-Karte muss eingelegt und initialisiert sein.
- Wird die Dateigröße durch die Schreibeoperation verändert, so wird diese möglicherweise nicht mehr korrekt am PC angezeigt.
- Bevor diese Funktion verwendet werden kann, muss `SL2_Init()` aufgerufen werden.

Beispiel

Liest Daten aus der Datei „test.c“ im Root-Directory der SD-Karte, speichert diese im RAM zwischen und überschreibt die Datei auf der SD-Karte mit eigenen Daten. Die auf die SD-Karte geschriebenen Daten werden wieder mit den Ausgangsdaten verglichen und die Datei wird dann wiederhergestellt.

```
int main ()
{
    uint8_t  Data[512] = "this␣is␣a␣simple␣test␣for␣the␣SD-card";
    uint8_t  DataBackup[512];
    uint8_t  DataReadback[512];
    uint16_t ClusterVar = 0;
    uint8_t  DirAttrib = 0;
    uint32_t Size = 0;
    uint8_t  ClusterSize = 0;
    uint16_t BytesPerSec = 0;
```

```
uint16_t Cnt = 0;
uint8_t  Temp = 0;

//function that has to be executed in every application to
//initialize the interface with the FPGA
Init();

//initialize the USB port to send debug information to the PC
TextInterfaceInit(38400);

//initialize the memory card
//this has to be done here AND before the memory card is
//used because otherwise there occur problems with various
//cards at power on
do
{
    Temp = mmc_init();
} while ( Temp != SUCCESSFUL);

//clear all LEDs again
SetLedClearAll();

//wait until the key is released and pressed again
AvrWaitOnKey (KEY_1, KEY_NOT_PRESSED);
AvrWaitOnKey (KEY_1, KEY_PRESSED);


//initialize the memory card again
do
{
    Temp = mmc_init();
} while ( Temp != SUCCESSFUL);

printf("accessing␣memory␣card...\n");

//save cluster OFFSET und cluster size
fat_cluster_data_store();

//get cluster size and bytes per sector
ClusterSize = GetClusterSize();
BytesPerSec = GetBytesPerSec();

//print some more data on the screen
printf("reading␣cluster␣size:␣%d\n", ClusterSize);
printf("reading␣bytes␣per␣sector:␣%u\n", BytesPerSec);
printf("testing␣memory␣card...␣");

//save the current content of the file in RAM,
//write into the file "test.txt" on the SD-card, read the
//data back, compare both and restore the file
if (fat_search_file("test.txt", &ClusterVar, &Size, &DirAttrib,
    DataReadback) == SUCCESSFUL)
{
    fat_read_file (ClusterVar, DataBackup, 0);

    fat_write_file (ClusterVar, Data, 0);
```

```
            fat_read_file (ClusterVar, DataReadback, 0);

            //compare bytewise
            for (Cnt=0; Cnt<512; Cnt++)
            {
                if (Data[Cnt] != DataReadback[Cnt])
                {
                    SignalizeError();
                }
            }

            fat_write_file (ClusterVar, DataBackup, 0);
            printf("OK\n");
        }
        else
        {
            printf("FILE NOT FOUND");
        }

        while(1)
        {
        }

        return 0;
}
```

9.11 Strukturen und benutzerdefinierte Typdeklarationen

9.11.1 Robotics-Chip

Servolinearisierung

Statt De nitionen werden für die Auswahl der Servo-Modi sowie des Reverse Bits die folgenden Enumerationstypen verwendet:

```
typedef enum
{
    RC_SERVO_MODE_NO_ACTION,
    RC_SERVO_MODE_PWM_DIRECT,
    RC_SERVO_MODE_PHI_SPEED,
    RC_SERVO_MODE_PHI_DIRECT
} RC_ServoMode;


typedef enum
{
    RC_SERVO_REVERSE_ON,
    RC_SERVO_REVERSE_OFF
} RC_ServoReverse;
```

Um bei Änderungen an der Hardware (z.B. bei Verwendung eines anderen Hexapods) möglichst wenig Änderungen in der Software durchführen zu müssen, sind alle hardwarespezi schen Werte in einer einzigen Datenstruktur abgelegt. Diese Struktur beinhaltet die Linearisierungswerte aller Servos,

den Offset-Winkel durch die Servoscheibe und das Reverse Bit (die Drehrichtung ist nicht bei allen Servomodellen gleich).

```
typedef struct
{
    uint16_t LinTable[RC_NUMBER_OF_SERVO_LIN];
    int16_t PhiOffset;
    RC_ServoReverse Rotation;
} RC_ServoAttributes;
```

PS/2 Maus

Die Struktur `RC_Ps2Data` enthält Daten über die Position und Tastenzustände der Maus. Eine genaue Beschreibung der einzelnen Registerwerte be ndet sich in Abschnitt 7.14 auf Seite 113.

```
typedef struct
{
    uint8_t DeltaX;
    uint8_t DeltaY;
    uint8_t Buttons;
} RC_Ps2Data;
```

Sony PlayStation 2 Gamepad

Die X- und Y-Koordinaten des Sony PlayStation 2 Gamepads sind in der Struktur `RC_GamePadJoystickData` zusammengefasst. Genauere Informationen über die Register für die Koordinaten nden sich in Abschnitt 7.16 auf Seite 115.

```
typedef struct
{
    uint8_t  X;
    uint8_t  Y;
} RC_GamePadJoystickData;
```

Um alle Informationen des Sony PlayStation 2 Gamepads lesen zu können existiert die Struktur `RC_GamePadData`. Darin sind zwei Strukturen vom Typ `RC_GamePadJoystickData` für die beiden Joysticks und eine 16-Bit Variable für die Tasten enthalten. Die Belegung, welche Taste welches Bit in `Buttons` beeinflusst, ndet sich in Abschnitt 7.16 auf Seite 115.

```
typedef struct
{
    uint16_t Buttons;
    RC_GamePadJoystickData JoystickLeft;
    RC_GamePadJoystickData JoystickRight;
} RC_GamePadData;
```

9.11.2 Inverse Kinematik

Für die Berechnung der inversen Kinematik ist ein dreidimensionales Koordinatensystem nötig, aus dessen Koordinaten die Winkel für die Servos berechnet werden.

```
typedef struct IK_Point3d
{
    int16_t X;
```

```
    int16_t Y;
    int16_t Z;
} IK_Point3d;
```

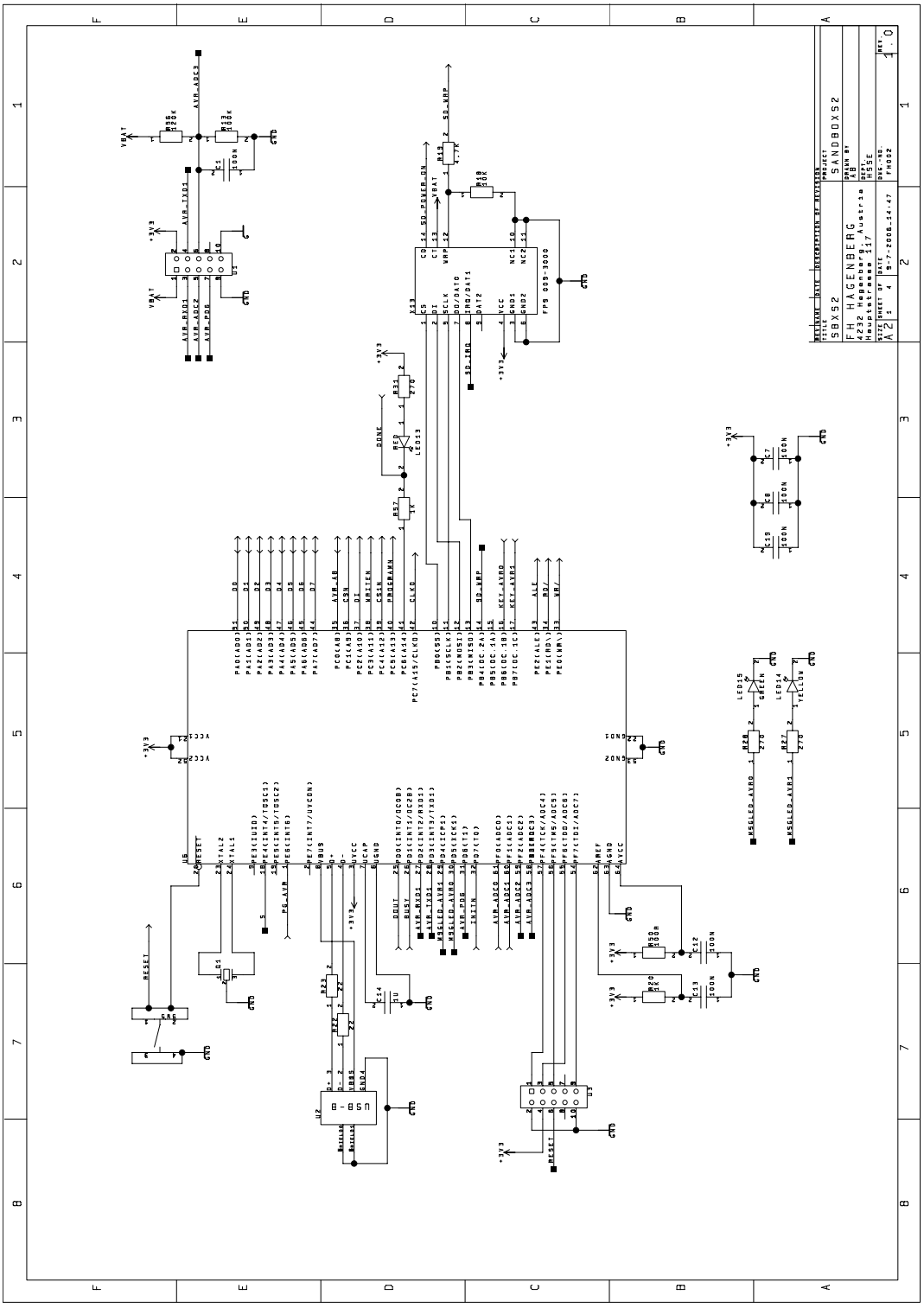
Weil die Höhenkoordinate bei der Drehung der einzelnen Koordinatensysteme in eine einheitliche Richtung nicht verändert wird, existiert ein zusätzlicher Datentyp für zweidimensionale Koordinaten.

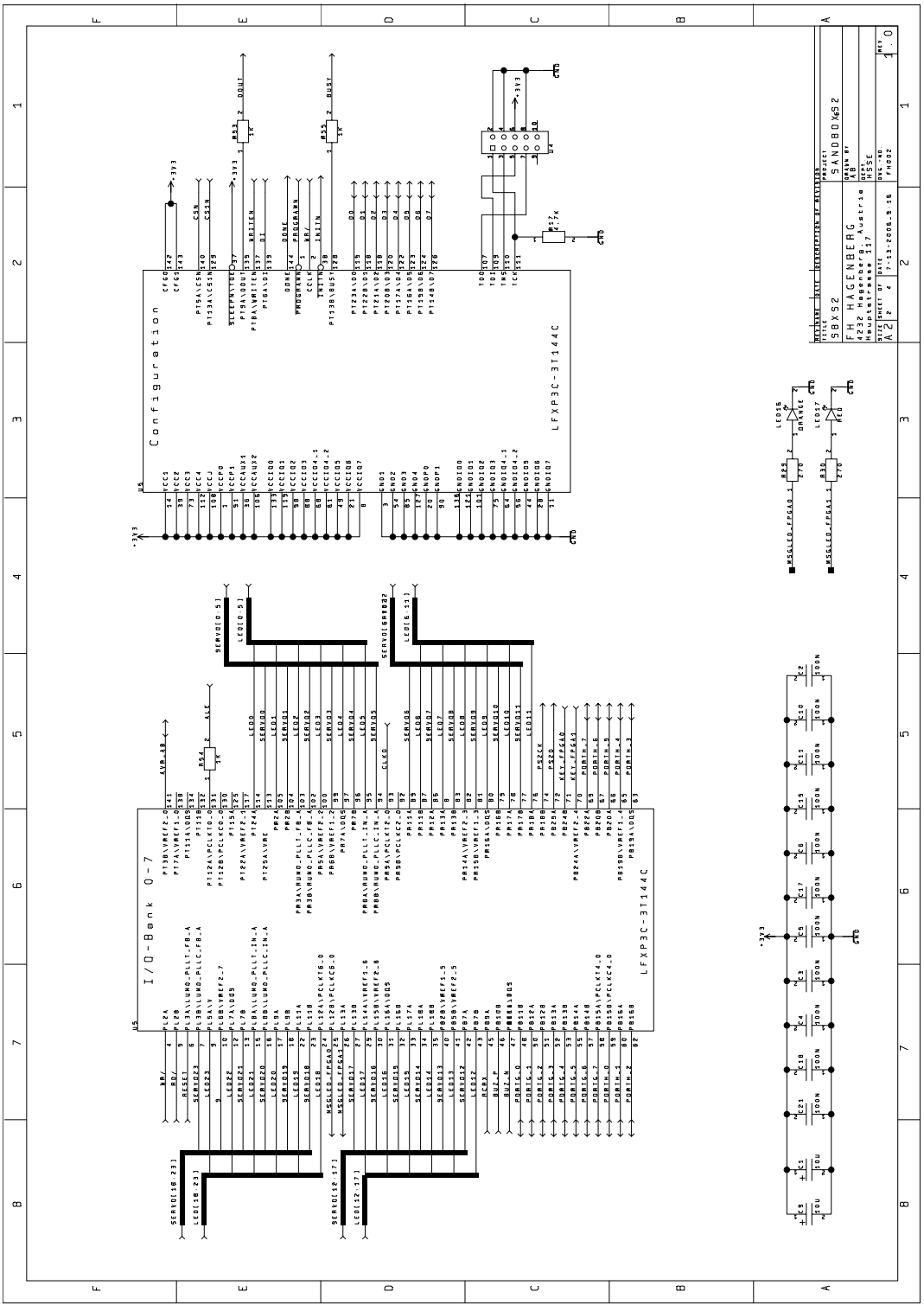
```
typedef struct IK_Point2d
{
    int16_t x;
    int16_t y;
} IK_Point2d;
```

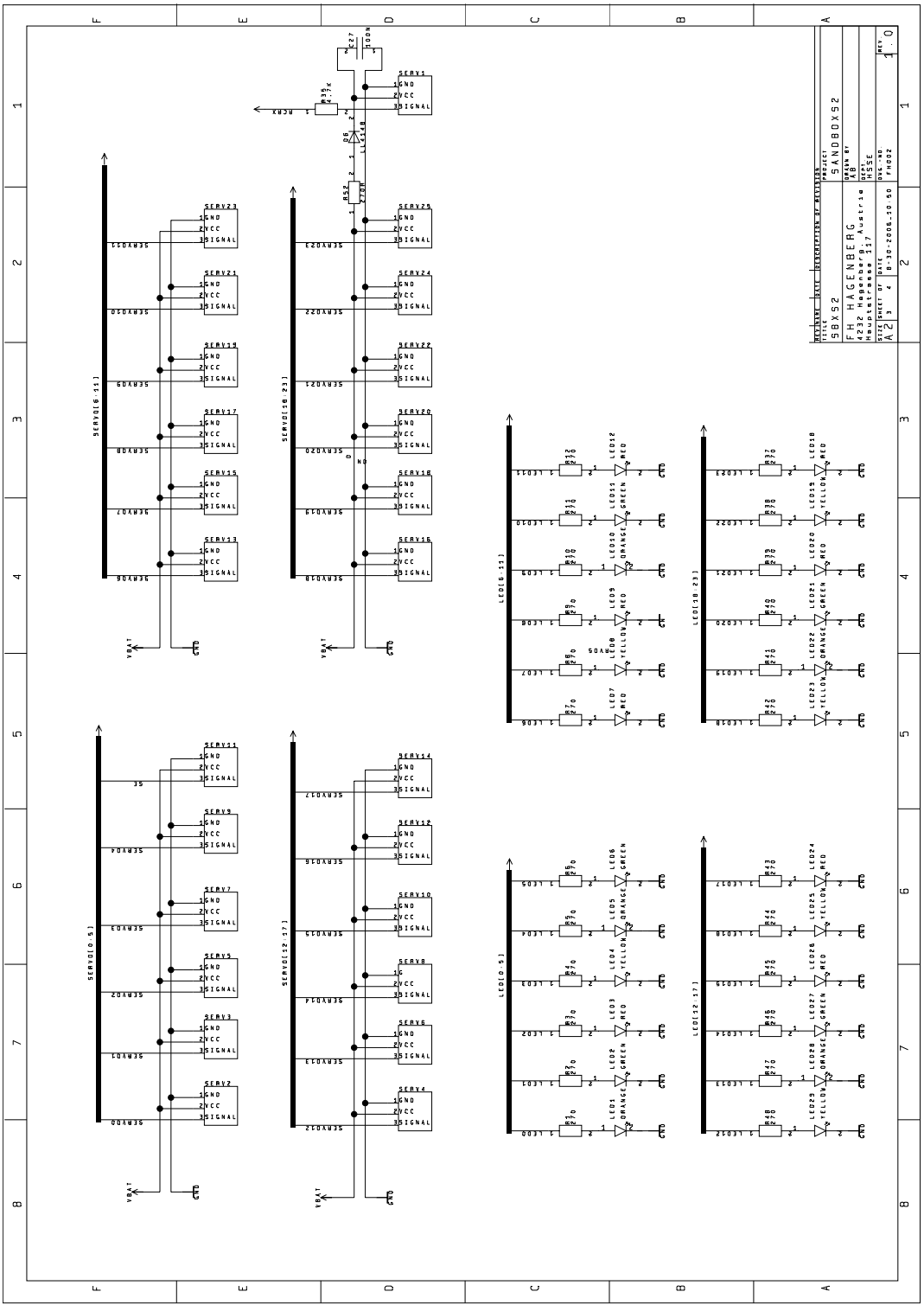
Die Winkel aller Servos für ein Bein sind in einer Struktur abgelegt und zusätzlich mit einem Statusbyte `Calculation` versehen, falls es sich um das Berechnungsergebnis der inversen Kinematik handelt.

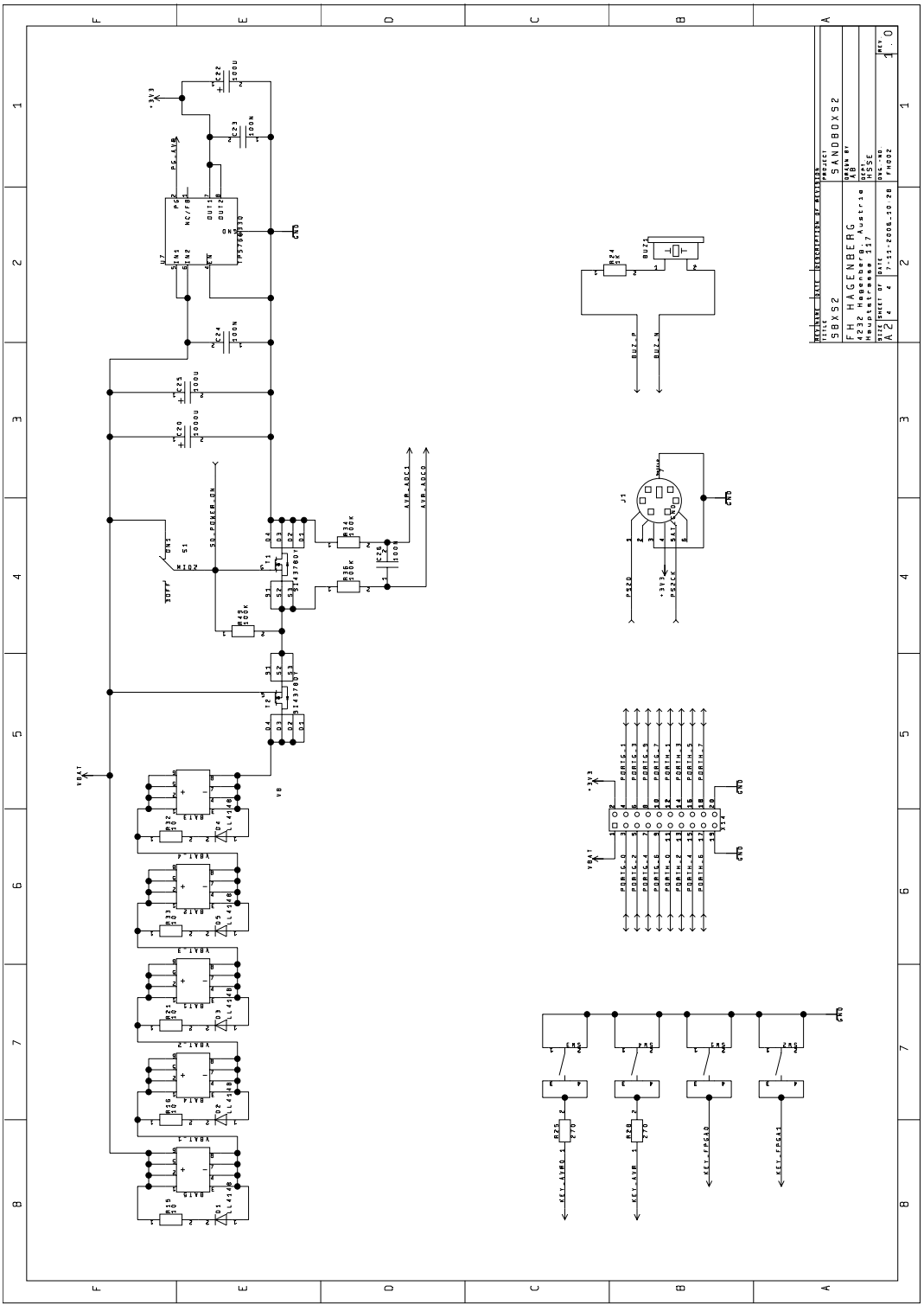
```
typedef struct IK_HexapodLegAngles
{
    int16_t Phi0;
    int16_t Phi1;
    int16_t Phi2;
} IK_HexapodLegAngles;


typedef struct IK_InverseKinematicsReturn
{
    IK_HexapodLegAngles Angles;
    uint8_t Calculation;
} IK_InverseKinematicsReturn;
```

A Schaltplan



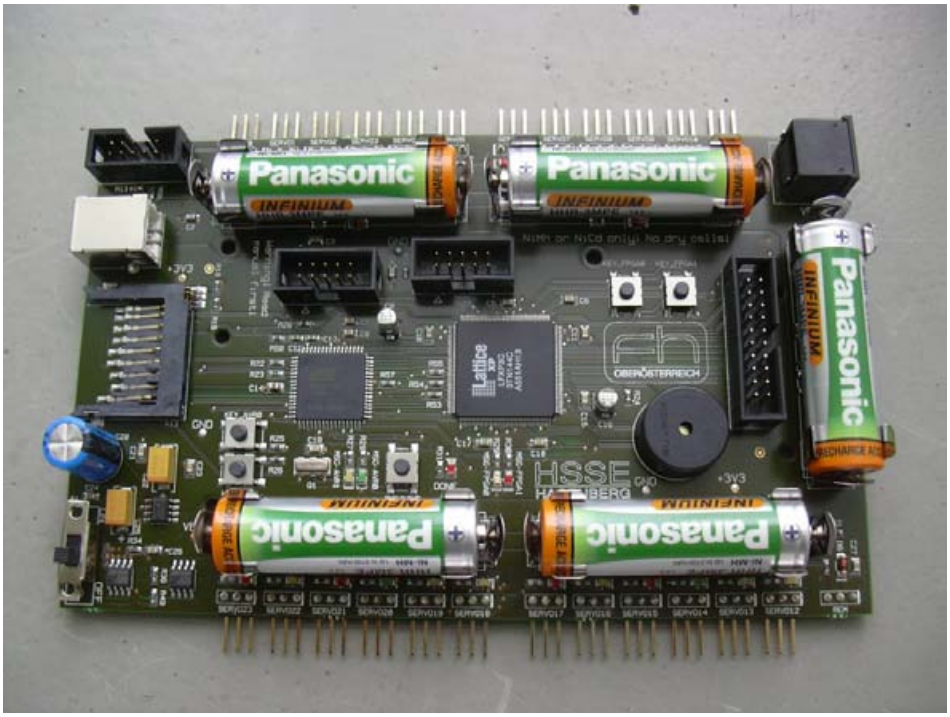

SBXS2
FH HAGENBERG
LFXP3C-3T144C
Configuration
I/O-Bank 0-7


SBXS2
SANDBOXS2
FH HAGENBERG
4232 Hagenberg
SERVO[0..5]
SERVO[6..11]
SERVO[12..17]
SERVO[18..23]
LED[0..5]
LED[6..11]
LED[12..17]
LED[18..23]
SERV1
SERV2
SERV3
SERV4
SERV5
SERV6
SERV7
SERV8
SERV9
SERV10
SERV11
SERV12
SERV13
SERV14
SERV15
SERV16
SERV17
SERV18
SERV19
SERV20
SERV21
SERV22
SERV23
SERV24
SERV25
GND
VCC
SIGNAL
VBAT
RCRX
LED1
LED2
LED3
LED4
LED5
LED6
LED7
LED8
LED9
LED10
LED11
LED12
LED13
LED14
LED15
LED16
LED17
LED18
LED19
LED20
LED21
LED22
LED23
LED24
LED25
LED26
LED27
LED28
LED29
GREEN
ORANGE
YELLOW
RED

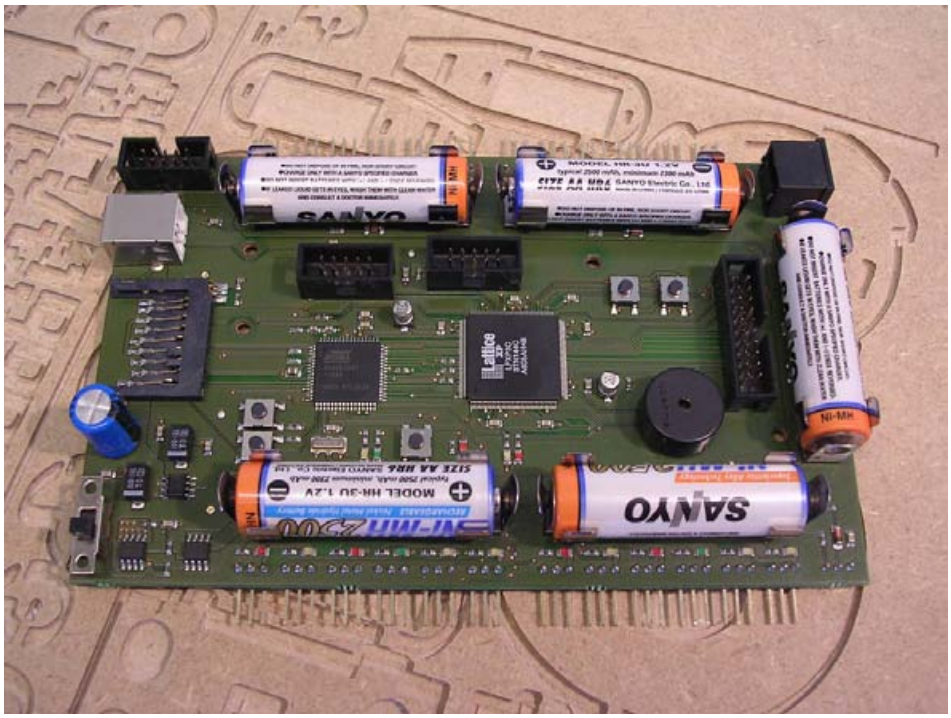


B Platinenlayout

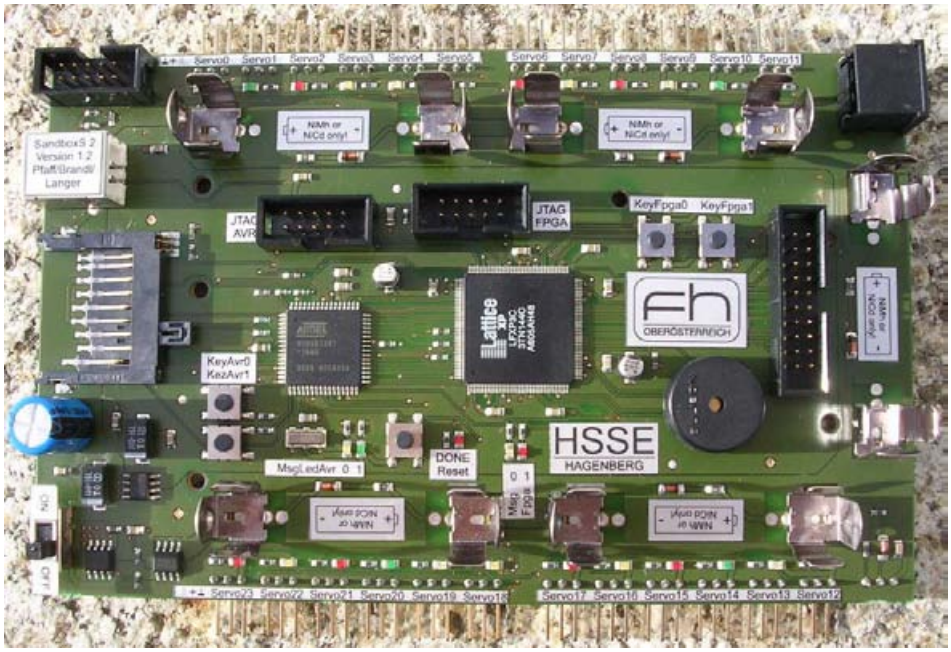
B.1 Platinenfoto ohne Bestückungsdruck



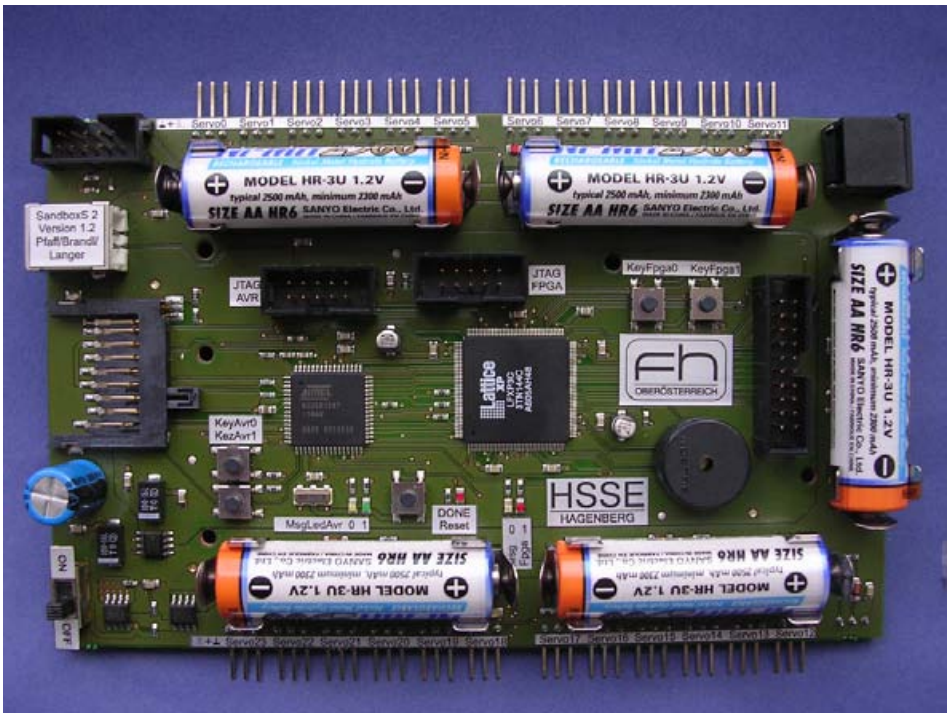
B.2 Platinenfoto ohne Bestückungsdruck



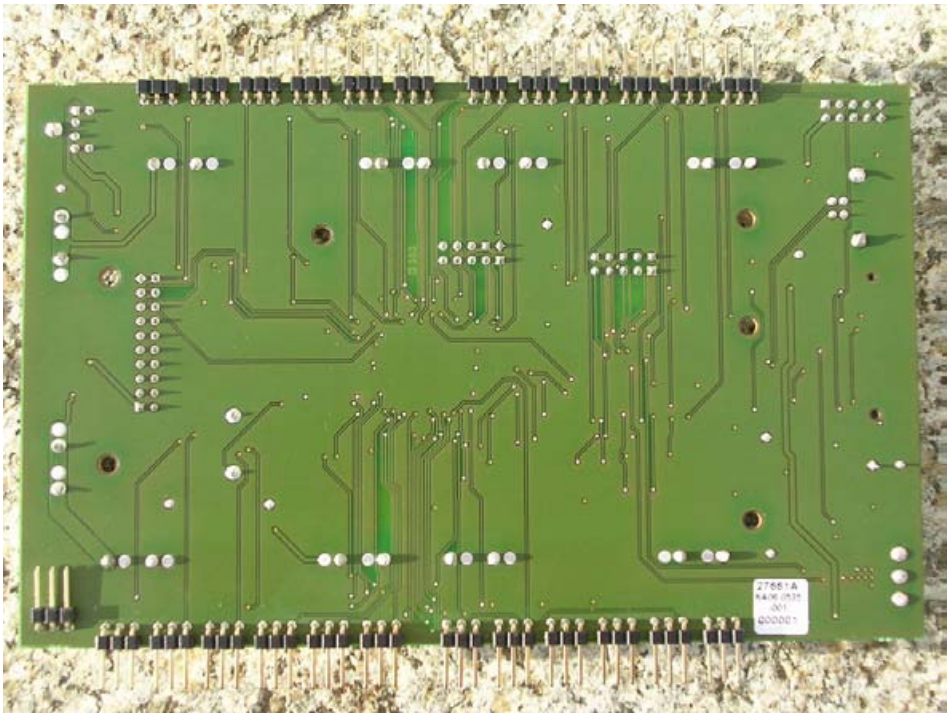
B.3 Platinenfoto mit Aufklebern



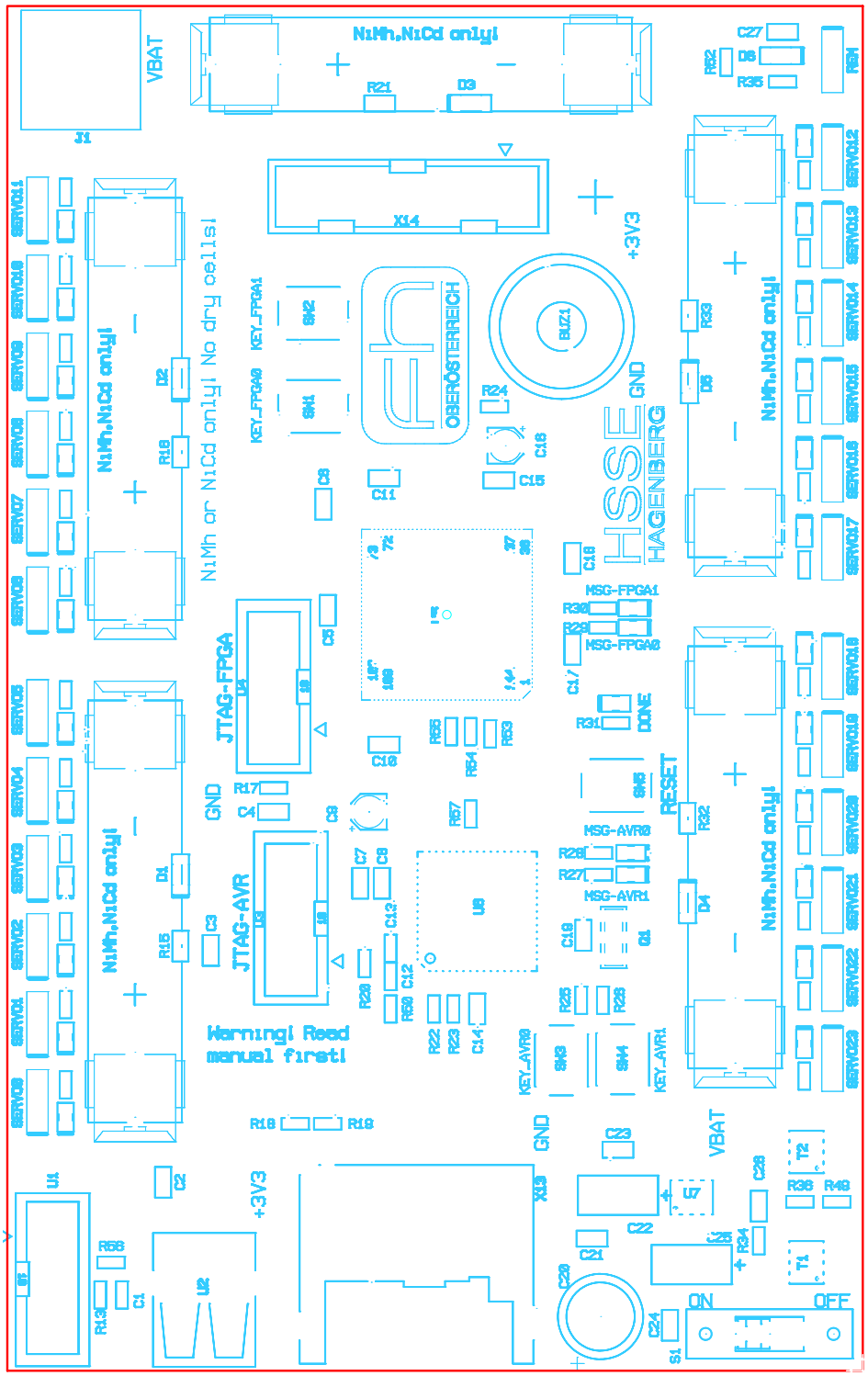
B.4 Platinenfoto mit Aufklebern und Akkuzellen



B.5 Foto der Platinenunterseite

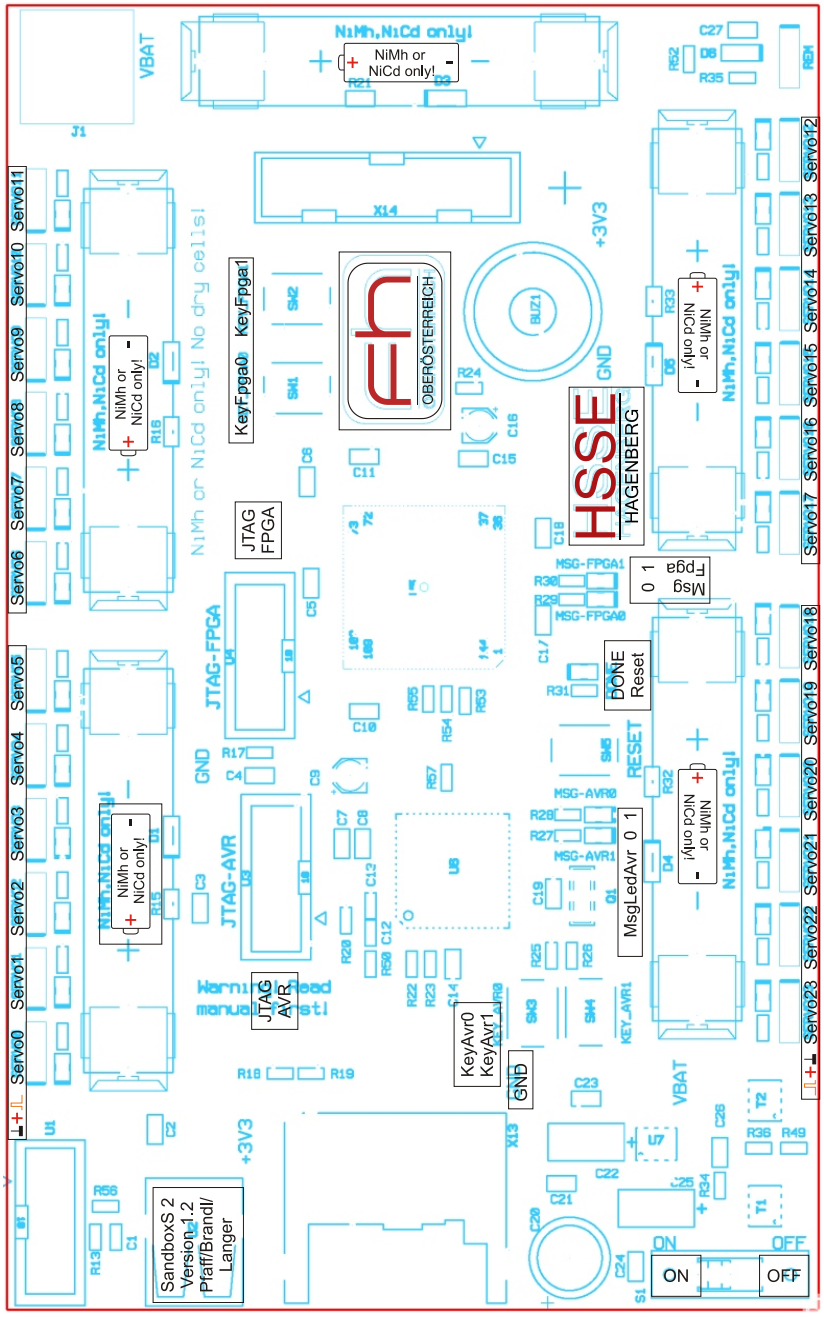


B.6 Bestückungsaufdruck

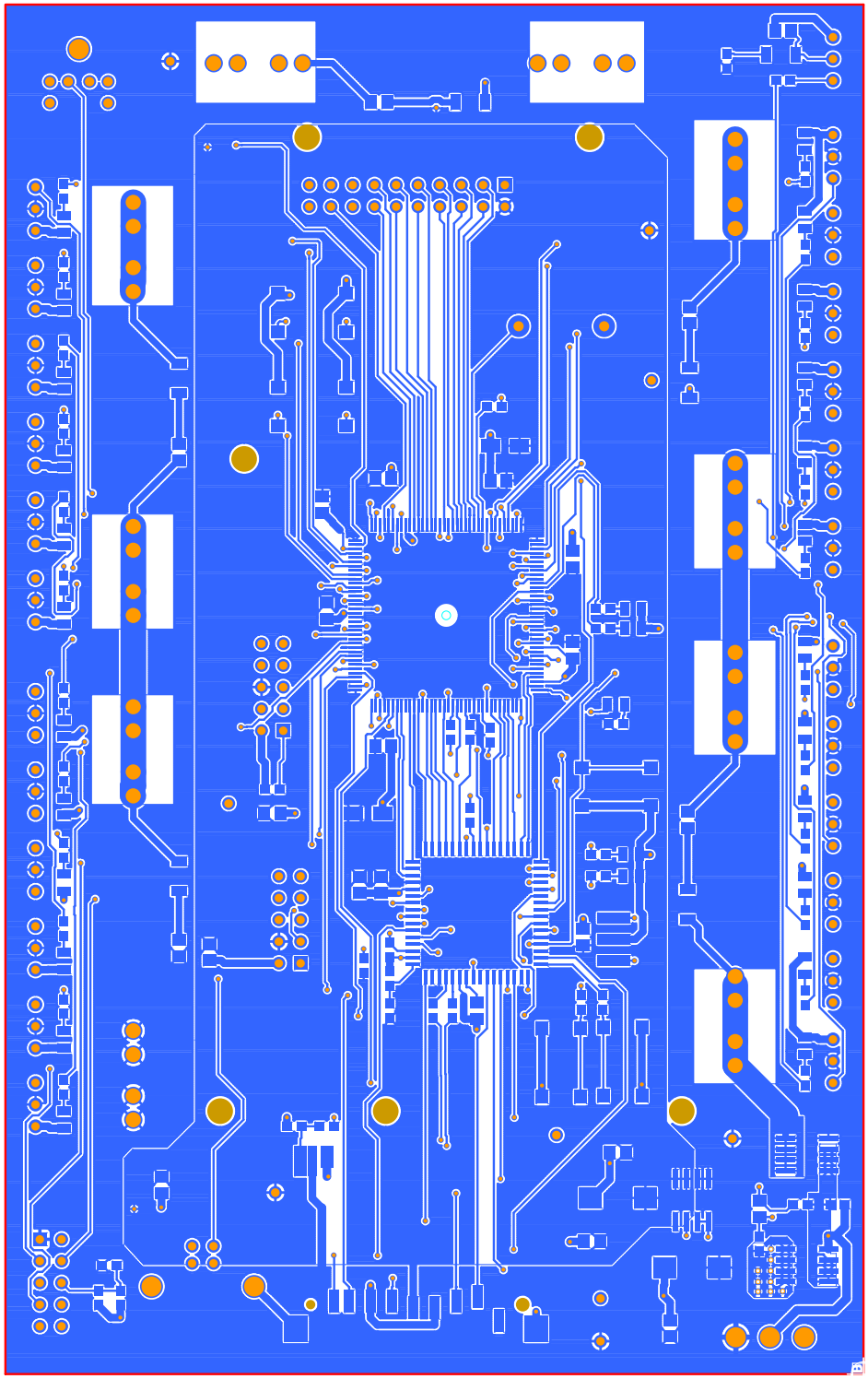


B.7 Aufkleber statt Bestückungsaufdruck

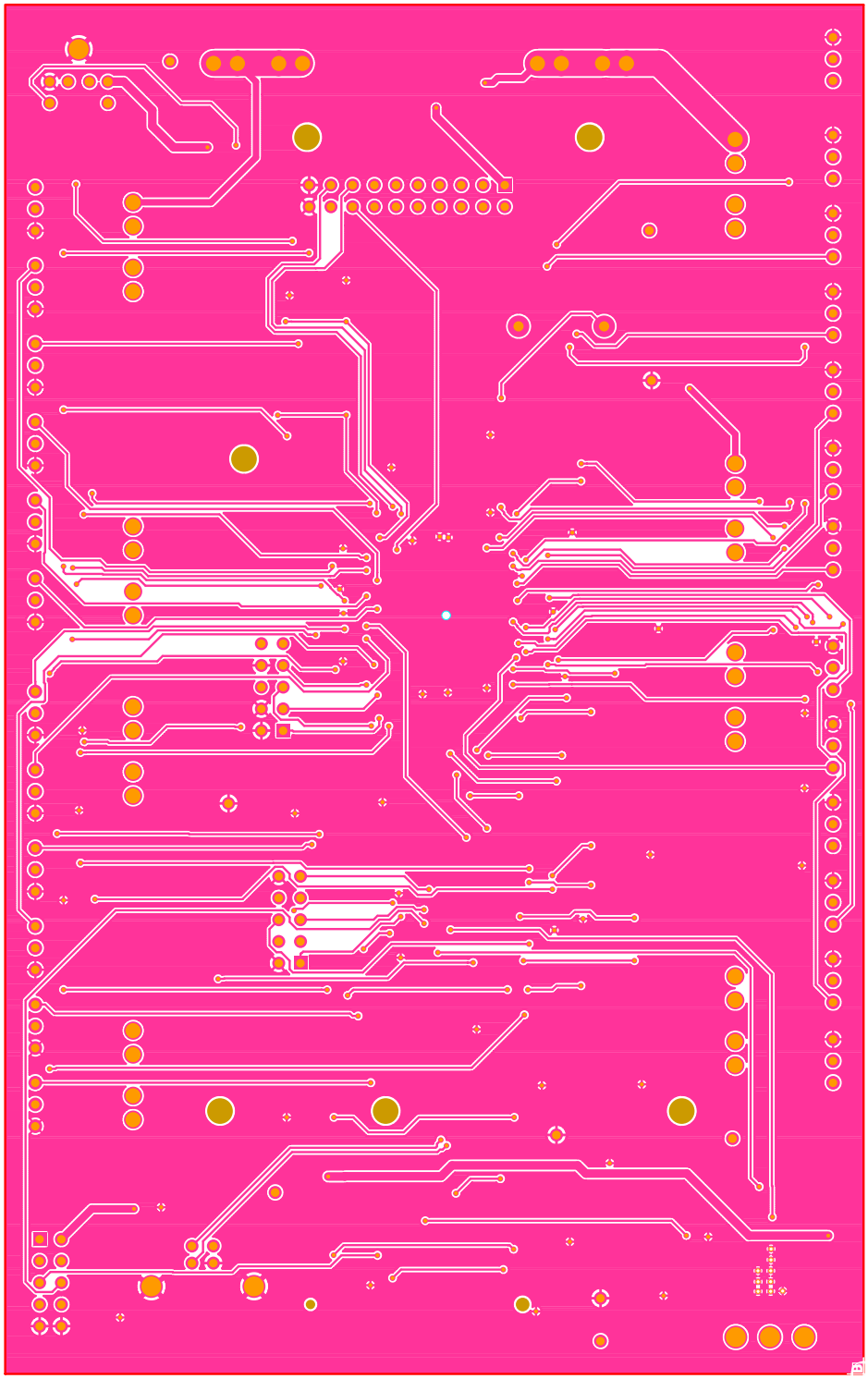
Alternativ zu einem Bestückungsdruck ist eine Beschriftung mit Aufklebern möglich. Die Abbildung gibt die Position der Aufkleber an.



B.8 Layout Top-Layer



B.9 Layout Bottom-Layer



C RC-Empfängerumbau zum Anschluss an die SandboxS 2

Prinzipiell kann so gut wie jeder RC-Empfänger so umgebaut werden, dass das PPM-Signal der Demodulationsstufe am Ausgang zur Verfügung steht. Als Beispiel ist die Modi kation eine preisgünstigen AM-Empfängers der Marke Acoms Techniplus beschrieben. Der Umbau von Empfängern anderer Fabrikate erfolgt sinngemäß, erfordert aber unter Umständen weitergehende Modi kationen als nachfolgend dargestellt.

Der unmodi zierte Empfänger ist links in Abb. C.1 zu sehen. Um die Modi kationen durchführen zu können, muss zunächst einmal der Schwingquarz entfernt werden (Abb. C.1, rechts). Hierzu wird dieser einfach herausgezogen. Vorsicht! Quarze sind erschütterungsemfpindliche elektro*mechanische* Bauteile. Keinesfalls fallen lassen!

Abbildung C.1: Links: Empfänger im Lieferzustand. Rechts: Der Quarz wurde entfernt.

Nun wird das Gehäuse geöffnet. Hierzu werden die beiden Schrauben an der Gehäuseunterseite herausgedreht (Abb. C.2, links). Der passende Schraubendreher trägt eine Klinge des Typs „Philips 0“.

Im Anschluss daran, kann der Empfänger in seine Einzelteile zerlegt werden. Beim Zerlegen gehen Sie bitte behutsam mit dem Plastikgehäuse um, da dieses bei zu hohem Kraftaufwand leicht Risse bekommt.

Abb.C.2 zeigt die Komponenten des zerlegten Empfängers.

Die Modi kationen werden ausschließlich auf der Platine vorgenommen. Gehäuse, Quarz und Schrauben in der Zwischenzeit also sicher verwahren, damit diese nicht verloren gehen.

Zuerst muss die 3-polige Stiftleiste des BATT-Anschlusses (Abb. C.3, links) ausgelötet werden.

Ein Lötkolben mit guter Heizleistung, Flussmittel, eine Elektronikzange und ein kleiner Schraubstock mit auf die Backen aufgesetzten Gummi- oder Filzschonern (z.B. Kugelgelenkschraubstock des

Abbildung C.2: Links: Gehäuseboden und passender Schraubendreher. Rechts: Einzelteile des zerlegten Empfängers.

Herstellers Proxxon) sind die für diesen Schritt ausreichenden Hilfsmittel. Zuerst spannen Sie die Platine — daher die Gummi- oder Filzschoner — in den Schraubstock. Bitte keine Gewalt anwenden, da dies zur mechanischen Zerstörung der Leiterplatte führen würde. Dann tragen Sie ein Flussmittel ausreichend dosiert auf die drei Lötpads der Stiftleiste auf. Anschließend wird Lötpad für Lötpad mit dem Lötkolben erwärmt und die einzelnen Stifte werden — wieder ohne Gewaltanwendung — mit einer Zange herausgezogen (Abb. C.3, rechts). Die Zange kühlt den Stift und damit die Lötstelle stark ab. Daher sollte man den Stift erst greifen, wenn das Lot flüssig geworden ist und dann unmittelbar herausziehen.

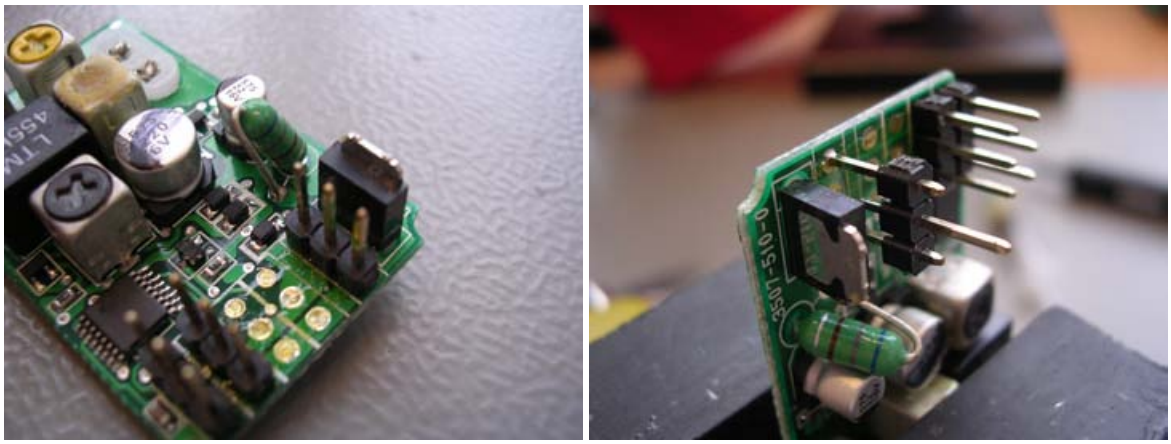


Abbildung C.3: Links: Stiftleise des BATT-Anschlusses. Rechts: Stift für Stift wird die Stiftleise ausgelötet.

Nachdem die Stifte entlötet sind, wird mit einer Entlötsaugpumpe das restliche Altlot aus den Bauteillöchern entfernt. In Abb. C.4 ist das gewünschte Ergebnis zu sehen.

An die Stelle wo sich die Stiftleiste befand, wird nun ein handelsübliches Servokabel angelötet (Abb. C.5). Hierbei die Farben der einzelnen Adern nicht vertauschen: Schwarz an GND (außen), rot an die Versorgungsspannung (Mitte), weiß an den freien Anschluss. Die Farbkodierung des Signalan-

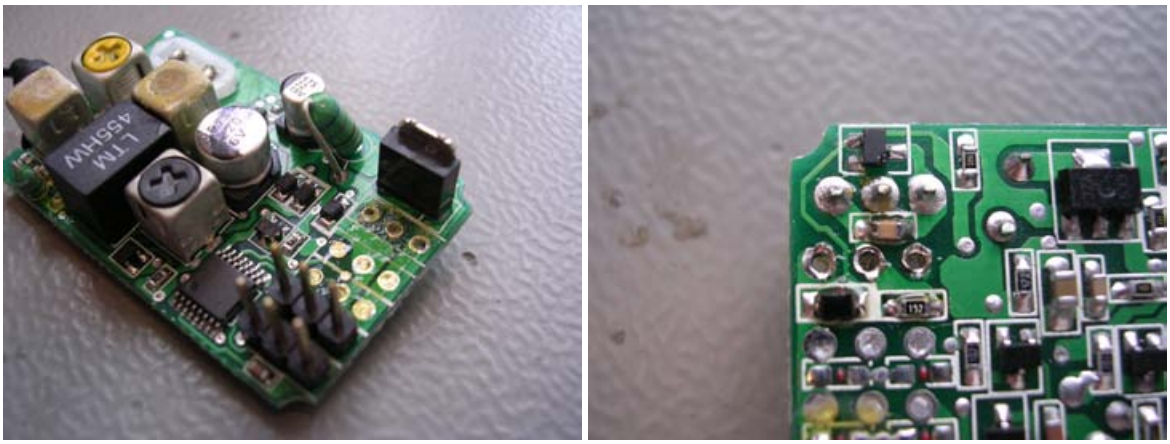


Abbildung C.4: Links: Die BATT-Stiftleiste ist vollständig entfernt. Rechts: Das Altlot wurde ebenfalls entfernt.

schlusses unterscheidet sich für unterschiedliche Servohersteller. Bei Hitec ndet sich hier beispielsweise eine gelbe Ader.

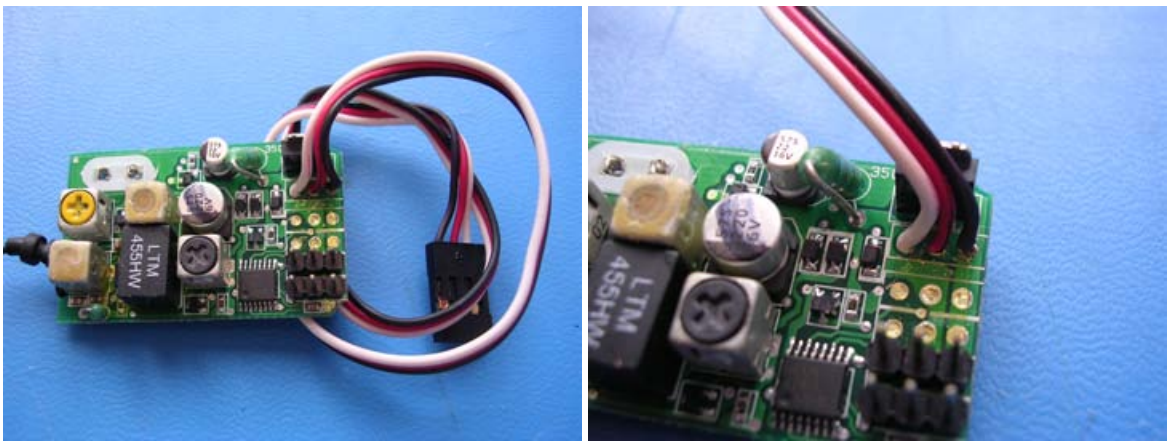


Abbildung C.5: Das an Stelle der BATT-Steckleiste angelötete Servoanschlusskabel.

Das für die Steuerung des Hexapods benötigte PPM-Signal muss auf der weißen Ader des Servokabels verfügbar sein. Im jetzigen Zustand ist diese Ader "not connected". Um das PPM-Signal an die Ader anzuschließen muss dieses mittels eines Oszilloskops auf der Platine aufgespürt werden (siehe hierzu Abschnitt 7.13). Im Falle des Acom-Empfängers ist dies bereits geschehen, so dass eine Drahtbrücke laut Abb. C.6 das gewünschte Ergebnis erzielen wird. Am Besten verwendet man hierfür ein Stück Wire-Wrap-, oder Kupferlack-Draht.

Nun kann der Empfänger wieder in sein Gehäuse zurück. Zu beachten ist, dass der Servokabelstecker wie links in Abb. **??** zu sehen, von innen durch den Gehäusedeckel geführt werden muss. Wird ein Stecker verwendet der nicht durch die Öffnung passt (teilweise beim Stecksystem von Futaba der Fall), dann muss das Servokabel bereits vor dem Einlöten durch diese Öffnung gezogen werderden.

Anschließend wird der Empfänger wieder komplett zusammengebaut und verschraubt(Abb. C.7, rechts).

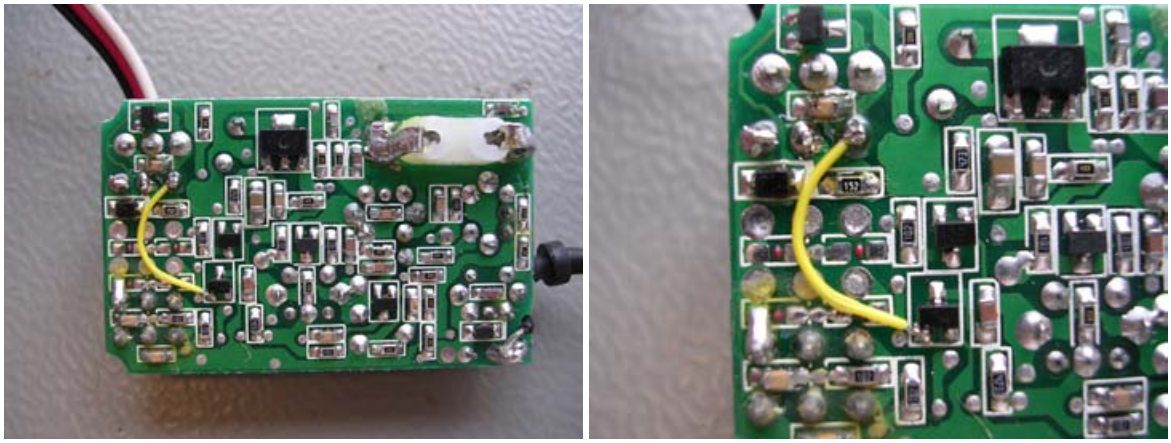


Abbildung C.6: Die Drahtbrücke legt das PPM-Signal am Ausgang der Empfängerdemodulationsstufe auf die Signalader (im Foto weiß) des zuvor eingelöteten Servokabels.

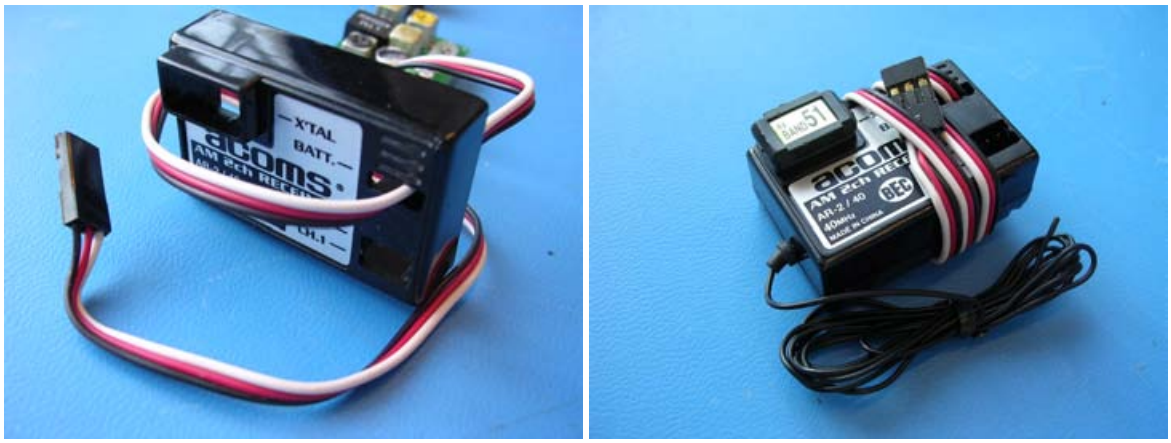


Abbildung C.7: Der modi zierte Empfänger ist einbaufertig. Im Hintergrund des linken Fotos wartet bereits die Platine eines weiteren Empfängers auf die Modi kation.

D Literaturverzeichnis

Craig, John J.: Introduction to Robotics: Mechanics and Control. 3. Auflage. Englewood Cliffs, New Jersey: Prentice-Hall, 2004

Kreuzgruber, Michael: CORDIC-Algorithmus. Bachelorarbeit FH-OÖ/Hagenberg, Hardware/Software Systems Engineering, Hagenberg, Austria, 2007

Oehme, Friedrich, Huemer, Mario und Pfaff, Markus: Elektronik und Schaltungstechnik — Ein verständlicher Einstieg. 1. Auflage. München: Carl Hanser Verlag, Oktober 2006, ISBN 978-3-446-40694-0

Dieses Buch soll für technisch interessierte Leser den Zugang zu elektronischen Schaltungen eröffnen, ohne dass ein Grundlagenstudium der Elektrotechnik vorausgesetzt wird. Die Grundlagen werden, soweit benötigt, mit behandelt.

Society, IEEE Computer (Hrsg.): IEEE Standard Test Access Port and Boundary-Scan Architecture. New York: Institute of Electrical and Electronics Engineers, 1993

Spong, Mark W., Hutchinson, Seth und Vidyasagar, Mathukumalli: Robot Modeling and Control. New York: John Wiley & Sons, Inc., 2005

Volder, J. E.: The CORDIC Trigonometric Computing Technique. In IRE Trans. on Electronics Computers, vol. EC-8, no. 3. IRE Trans, September 1959, 330–334

Walther, J. S.: A Uni ed Algorithm for Elementary Functions. In AFIPS 1971. Atlantic City, USA: Proc. Spring Joint Computer Conf, 1971, 379–385

Index

External Memory Interface, 57

Fuse, 57

Haftungsausschluss, 2

JTAG-ICE, 58

Legende, 2
Linearisierung, 91

Modellbauservo, 87

Nutzungsbedingungen, 2

Raumkoordinate, 91
Regelkreis, 89
Robotikservo, 87, 89

Servo, 87
Servokennlinie, 91
Servostecker, 92

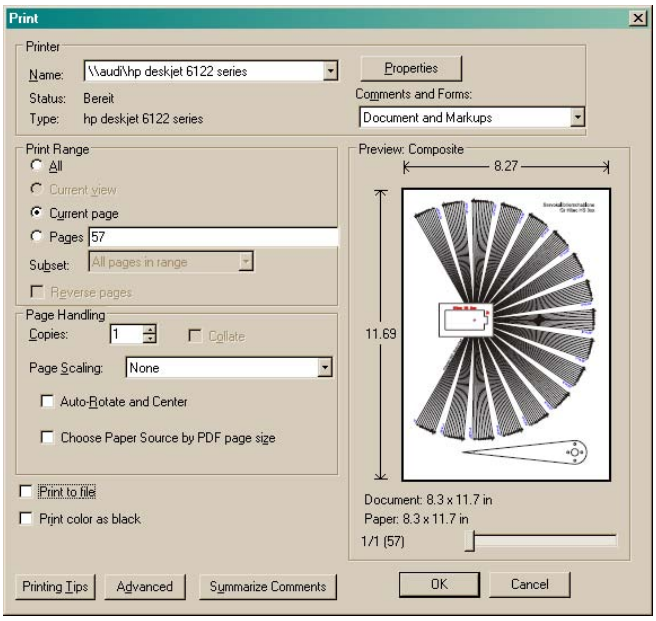
Zeichenlegende, 2

Servokalibrierschablone
für Hitec HS 3xx

Copyright 2007, Markus Pfaff

Ausdrucken der Winkelkalibrierschablone

Die umseitige Winkelkalibrierschablone muss unverzerrt ausgedruckt werden. Im Acrobat-Reader wird hierzu die Seite mit der Winkelschablone aufgeschlagen und mit den Einstellungen „Current page“ für den Parameter „Print Range“und „None“ für den Parameter „Page Handling“ ausgedruckt. Ein Beispiel für diese Einstellungen ist im nachfolgenden Screenshot zu sehen:



Beispiel für den Druckdialog von Acrobat-Reader. Die Einstellung für den Drucker muss selbstverständlich an die verwendete Computer-/Druckerkombination angepasst werden.